[redacted]

Math

.996 Report No. UIUCDCS-R-79-996

UILU-ENG 79 1745

DESIGN OF DIAGNOSABLE MOS NETWORKS

by

Hung Chi Lai

December 1979

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-79-996

DESIGN OF DIAGNOSABLE MOS NETWORKS

by

Hung Chi Lai

December 1979

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the National Science Foundation under Grant No. DCR73-03421 A01.

A STUDY OF CURRENT LOGIC DESIGN PROBLEMS:

PART I, DESIGN OF DIAGNOSABLE MOS NETWORKS;

PART II, MINIMUM NOR (NAND) NETWORKS FOR PARITY FUNCTIONS OF AN ARBITRARY NUMBER OF VARIABLES;

PART III, MINIMUM PARALLEL BINARY ADDERS WITH NOR (NAND) GATES AND THEIR EXTENSIONS TO NETWORKS CONSISTING OF CARRY-SAVE ADDERS

BY

HUNG CHI LAI

B.Eng., University of Tokyo, 1968
M.Eng., University of Tokyo, 1970

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1976

Urbana, Illinois

ACKNOWLEDGEMENT

The author wishes to thank his advisor, Professor S. Muroga, for his invaluable guidance during the preparation of this thesis, and also for his careful reading and constructive criticism of the original manuscript. The author also wishes to thank Mr. J. N. Culliney and Dr. K. Kinoshita for their helpful discussions and constructive comments. The author is grateful for the typing of the manuscript by Mrs. Cheryl A. Kopmann. The financial support of the Department of Computer Science and the National Science Foundation is also acknowledged.

PREFACE

One of the logic designer's goals is to design economical logic networks. With the rapid progress of technology in large-scale integration, new problems, such as the design of logic networks with MOS complex cells, the design of logic networks with programmable logic arrays, and the minimization of large logic networks arise. Some of these problems are unsolved old problems, such as the minimization of large logic networks, with new implications. This thesis deals with some of these logic design problems which arise in the design of economical networks for large-scale integrated circuits.

This thesis consists of three parts. Part I deals with the problem of designing diagnosable MOS networks. An algorithm is given which designs multi-level diagnosable (irredundant) MOS networks with a minimum number of MOS cells for a given single function. This algorithm is extended to the case of networks with multiple incompletely specified outputs although the minimality of the designed network in terms of the number of cells is not guaranteed for networks with multiple outputs. Some interesting diagnostic properties of the designed networks are also discussed.

Part II of this thesis discusses minimum NOR (or NAND) networks for parity functions of an arbitrary number of variables. Design procedures for networks with a minimum number of gates or a minimum number of connections for parity functions are given, and the minimality of these networks is proved by mathematical induction. The number of

variables can be arbitrarily large, and these networks are suitable for LSI implementation such as IIL (integrated injection logic) implementation.

Another study of minimum NOR (or NAND) networks of large sizes--minimum networks for n-bit parallel binary adders--is presented in Part III. A list of basic building blocks, obtained with the assistance of a computer program for designing minimum NOR networks based on the branch-and-bound method developed at the Department of Computer Science, University of Illinois, is given, and a procedure utilizing these basic networks to design parallel binary adders with a minimum number of NOR (or NAND) gates under the condition that only uncomplemented variables are available as inputs is also presented. These n-bit parallel adders consist of 7n+1 gates which is proved to be the lower bound on the number of gates required by an n-bit parallel adder. This design approach is extended to the case of networks which consists of carry-save adders (one-bit full adders). Those networks are suitable for high-speed realization of arithmatic operations involving repeated additions, such as multiplication and multi-operand addition. Several examples are presented in which the newly developed approach can save as much as 25% of the gates required by the conventional design.

For the sake of convenience, the topics presented in the three parts are presented independently. Each part is self-contained. References to sections are within the same part unless otherwise noted.

1. INTRODUCTION

With the recent progress in the technology of large-scale integration (LSI), MOS networks have been attracting more and more attention. Since an MOS cell theoretically can be made to realize an arbitrary negative function, an MOS cell is sometimes represented by a negative gate [Spe 69] and [IM 71]. Many studies have been carried out in recent years regarding the synthesis of negative gate networks. As a result of a recent investigation by T. Ibaraki and S. Muroga [IM 71], a method for designing two-level networks with a minimal number of negative gates was developed. T. Ibaraki [Iba 71] further extended this approach to the problem of minimizing the number of connections in a two-level negative gate network based on solving covering problems. These results were extended to the synthesis of multi-level networks with minimum numbers of negative gates by T. Nakamura, N. Tokura, and T. Kasami [NTK 72] based on a general structure of a feed-forward negative gate network. T. K. Liu [Liu 72] independently discovered a similar approach to design multi-level or two-level networks with a minimal number of negative gates. Although the two approaches differ in certain aspects, they are essentially based on the same type of grouping of input vectors. As a result, the networks designed by their approaches may contain redundant corrections and MOSFET's.

This part of the thesis will deal with the problem of designing "irredundant" multi-level MOS networks with a minimum number of MOS cells. Section 2 will introduce some symbols and definitions to be used in this part, and will explain the operation of a typical MOS cell to show why an MOS cell realizes a negative function. Section 3 will review previously obtained results on the synthesis of multi-level networks with a minimum number of negative gates. The similarity of the two approaches by [NTK 72] and [Liu 72] will be discussed. Variations of their algorithms will also be presented which will be used in a procedure, to be introduced in Section 6, for designing irredundant MOS networks. Based on these algorithms, the uniqueness of a network with a minimum number of negative gates for a given function will be explored in Section 4. Section 5 will be devoted to the problem of designing MOS cells for a negative function. The possible existence of redundant FETs in the MOS networks designed by known approaches will be discussed in detail. To solve this problem, a procedure for designing irredundant MOS networks with a minimum number of MOS cells will be presented in Section 6 and its validity will be proved. Some diagnostic properties of the networks designed by this procedure will be discussed in Section 7. The above discussions will be extended to the cases of networks with incompletely specified output functions and networks with multiple outputs in Sections 8 and 9, respectively. It will be shown that the approach discussed in Section 6 for designing irredundant MOS networks can be modified to be applied to the above cases also. However, in the case of networks with multiple outputs, the networks designed by this approach may not contain a minimum number of negative gates although they are guaranteed not to contain any redundant FETs.

2. DEFINITIONS

In this section, we will introduce symbols and definitions to facilitate our discussion. We will also present basic lemmas, theorems and corollaries related to negative functions which are mostly discussed in [IM 71], [NTK 72], and [Gil 54]. Consider switching functions of n variables $x_1,\ldots,x_n$. Let V_n be the set of all n-dimensional binary (0 or 1) vectors. Suppose two vectors $A = (a_1,\ldots,a_n) \in V_n$ and $B \in (b_1,\ldots,b_n) \in V_n$ are given. $a_i = b_i$ for every $i=1,\ldots,n$ is denoted by $A = B$, and $a_i \leq b_i$ for every $i=1,\ldots,n$ but $a_i < b_i$ for at least one i is denoted by $A < B$. If none of the relations $A > B$, $A < B$, and $A = B$ holds between A and B, then A and B are said to be <u>incomparable</u>. For convenience, let 0 denote $(0,\ldots,0) \in V_n$ and I denote $(1,\ldots,1) \in V_n$.

For a graphical representation, an n-dimensional cube C_n is defined as follows:

(1) Each vector in V_n is represented as a distinct vertex in C_n. Henceforth a vertex in C_n may be referred to by the vector which it represents;

(2) There is an edge between two vertices A and B if and only if A differs from B at exactly one component;

(3) The edge between A and B is directed from A to B if $A > B$ and is denoted by $\overrightarrow{AB}$.

Fig. 2.1 shows a three cube C_3.

The <u>weight</u> of a vertex A is defined as the number of "1" components in vector A.

Let $f_1,\ldots,f_m$ be completely specified switching functions of n variables. An n-cube C_n is referred to as a labeled n-cube with respect to functions $f_1,\ldots,f_m$ when a binary integer,

$$L(A) \equiv \ell(A;f_1,\ldots,f_m) = \sum_{i=1}^{m} f_i(A)\ 2^{m-i}$$

is attached to each vertex A of C_n as a label. A labeled n-cube with respect to $f_1,\ldots,f_m$ is denoted by $C_n(f_1,\ldots,f_m)$. Fig. 2.2 shows a labeled 3-cube with respect to functions $f_1 = \bar{x}_1 \vee \bar{x}_3$ and $f_2 = x_1x_3 \vee \bar{x}_1x_2\bar{x}_3$. Fig. 2.3 shows a labeled 3-cube with respect to the function $f = x_1x_2 \vee x_2\bar{x}_3$.

The following is the definitions related to negative functions which are based on the above graphical representations.

Definition 2.1 - A directed edge $\vec{AB}$ is said to be an inverse edge of a labeled n-cube $C_n(f_1,\ldots,f_m)$ if and only if $\ell(A;f_1,\ldots,f_m) > \ell(B;f_1,\ldots,f_m)$.

In the labeled 3-cube shown in Fig. 2.2, the directed edge $\overrightarrow{(010)(000)}$ in the bold line is the only inverse edge. In the labeled 3-cube for the single function $f = x_1x_2 \vee x_2\bar{x}_3$ (i.e., m=1) shown in Fig. 2.3, the directed edges, $\overrightarrow{(111)(101)}$, $\overrightarrow{(110)(100)}$, and $\overrightarrow{(010)(000)}$, shown in bold lines are inverse edges.

Based on the definition of inverse edges, a negative function can be defined as follows.

Definition 2.2 - A completely specified switching function f of n variables is a negative function if and only if there is no inverse edge in $C_n(f)$, the labeled n-cube with respect to f.

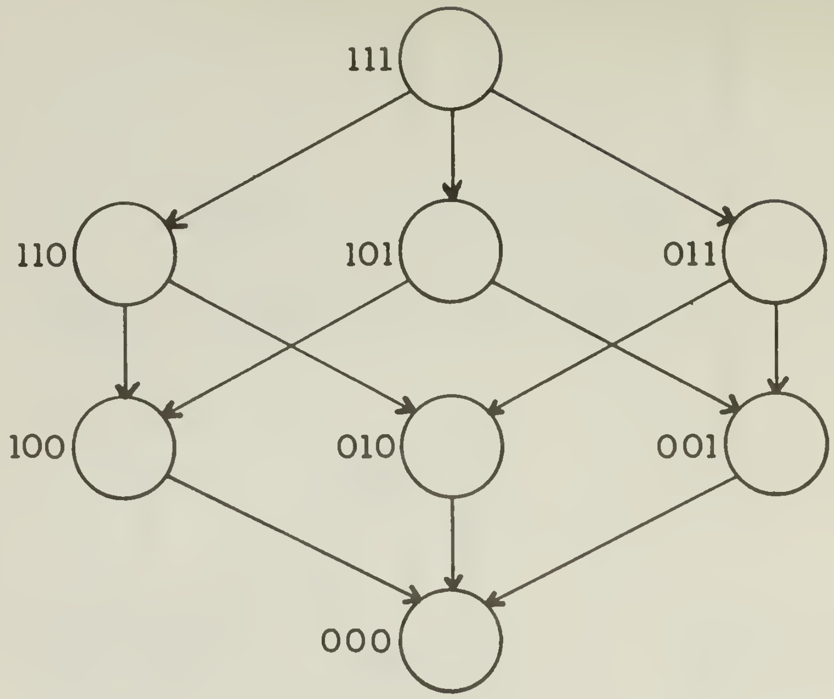


Fig. 2.1 3-cube C_3.

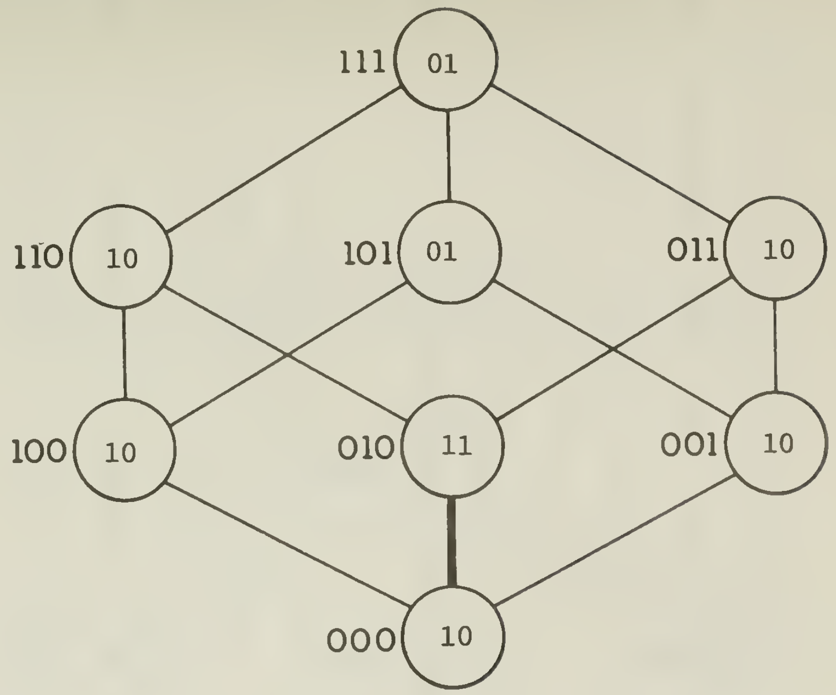


Fig. 2.2 A labeled 3-cube with respect to $f_1 = \bar{x}_1 \vee \bar{x}_3$ and $f_2 = x_1x_3 \vee \bar{x}_1x_2\bar{x}_3$, where the binary number inside each circle is the label attached to the corresponding vertex.

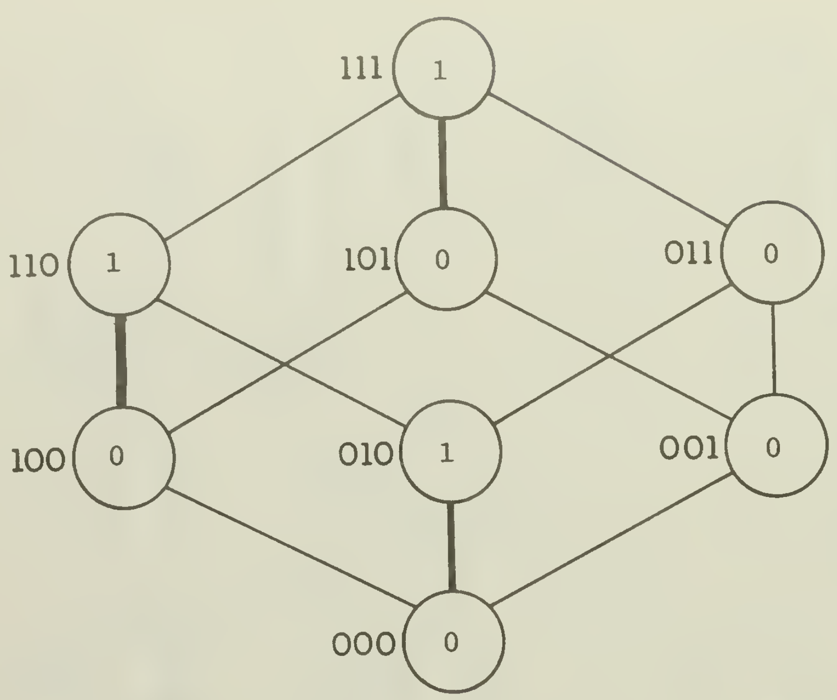


Fig. 2.3 A labeled 3-cube with respect to $f = x_1x_2 \vee x_2\bar{x}_3$, where the number inside each circle is the label attached to the corresponding vertex.

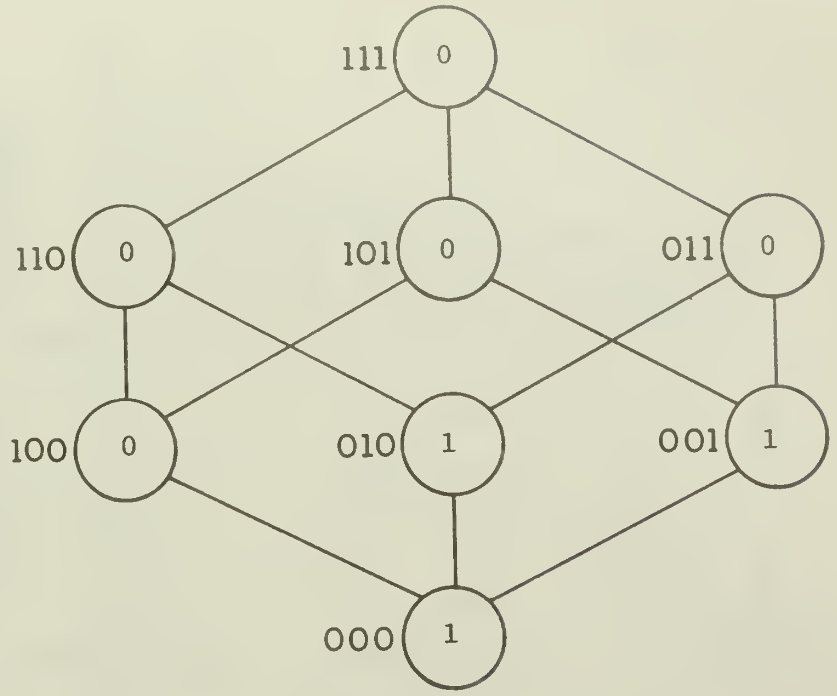


Fig. 2.4 A labeled 3-cube for $f = \bar{x}_1\bar{x}_3 \vee \bar{x}_1\bar{x}_2$ which has no inverse edge.

This definition is consistent with the conventional definition of negative function as can be seen from the following lemmas and theorems which state equivalent characterizations of negative functions. These characterizations can be found in [IM 71].

Lemma 2.1 - If a switching function f is a negative function, then $f(A) > f(B)$ does not hold for every pair of input vectors A, B such that $A > B$.

Proof - We only have to prove that f(B) must be 1 when f(A) is 1. Since f is a negative function, by definition, no directed edge $\vec{AB}$ of $C_n(f)$ can be an inverse edge. Therefore, $f(B)=1$ for every $B \varepsilon C_n$ such that $\vec{AB} \varepsilon C_n$. Repeating this argument, it is obvious that $f(B)=1$ for every $B \varepsilon C_n$ such that $A > B$. Hence, $f(B)=1$ must hold if $f(A)=1$ and $A > B$.

Q.E.D.

Corollary 2.1 - If a switching function f is a negative function and $f(A)=1$ for some $A \varepsilon C_n$, then $f(B)=1$ for every $B \varepsilon C_n$ such that $B < A$. Similarly, if $f(A)=0$ for some $A \varepsilon C_n$, then $f(B)=0$ for every $B \varepsilon C_n$ such that $B > A$.

From Lemma 2.1, the following theorem can be easily proved.

Theorem 2.1 - A function f of n variables is a negative function if and onlv if for every pair of input vectors such that $f(A)=1$ and $f(B)=0$ there exists a subscript i $(1 \leq i \leq n)$ such that $a_i=0$, $b_i=1$.

Proof - First let us prove the "if" part of this theorem. Suppose that for every pair of A and B such that $f(A)=1$ and $f(B)=0$ there exists an i such that $a_i=0$. $b_i=1$. We will prove that there is no inverse edge in $C_n(f)$. Suppose this is not true, in other words, there is a directed edge $\vec{AB}$ such that $f(A)=1$ and $f(B)=0$. For this particular pair of A and B, there will be no subscript i satisfying $a_i=0$ and $b_i=1$ because $\vec{AB}$ is a directed edge, and consequently $A > B$. This contradicts the assumption and therefore the "if" part of the theorem must hold.

Next let us prove the "only if" part of the theorem. Suppose f is a negative function and there is a pair of vectors A and B satisfying $f(A)=1$ and $f(B)=0$. We will prove that there must exist a subscript i such that $a_i=0$ and $b_i=1$. If $A < B$ or A is not comparable with B, then there must be some i such that $a_i=0$ and $b_i=1$. If $A > B$, then by Lemma 2.1 $f(A) \nsucc f(B)$ which contradicts the assumption that $f(A)=1$ and $f(B)=0$. Therefore, the "only if" part of this theorem must hold.

Q.E.D.

Because of this characterization, it is sometimes convenient to determine whether or not a function is a negative function by checking whether or not all 0-1 combinations in the values of the function have corresponding 1-0 combinations in the values of the variables. For convenience, let us use the term <u>0-1 pair</u> to refer to any combination of "0" and "1" in the values of the function, and use the term <u>1-0 cover</u> to refer to the corresponding combination of "1" and "0" in the values of a particular variable of this function. From Theorem 2.1, it is clear that function f is a negative function if and only if all 0-1 pairs in f are covered by 1-0 covers in its variables.

Another characterization of negative functions is given by the following theorem.

Theorem 2.2 - A function is a negative function if and only if it has a disjunctive form consisting of complemented variables only.

This theorem can be easily proved based on a theorem in [Gil 54] which characterizes positive functions.

Fig. 2.4 shows a labeled 3-cube for a negative function $f = \bar{x}_1\bar{x}_3 \vee \bar{x}_1\bar{x}_2 = \overline{x_1 \vee x_2x_3}$. Obviously, this labeled 3-cube has no inverse edge (Definition 2.2), and for every 0-1 pair in f, there is a 1-0 cover. For example, the 0-1 pair, $f(110) \equiv f(A)=0$ and $f(000) \equiv f(B)=1$, is covered by the 1-0 cover, $a_1=1$ and $b_1=0$, or $a_2=1$ and $b_2=0$.

The above definitions and discussions are concerned with completely specified functions; i.e., functions whose values are specified for every input vectors. For incompletely specified functions, i.e., functions whose values for some input vectors are not specified (don't cares), we need the following definitions and theorems. We use "*" to denote those unspecified values of a function.

Definition 2.3 - A *completion* of an incompletely specified function f is a completely specified function obtained from f by specifying each unspecified value of f to either "1" or "0." If the resulting completion is a negative function it is called a *negative completion*.

The negativeness of an incompletely specified function of n variables is defined as follows.

Definition 2.4 - An incompletely specified function f of n variables is *negative with respect to* these n-variables if and only if it has a negative completion.

The following theorem gives a criterion to judge whether or not an incompletely specified function is negative with respect to its input variables. For convenience, we sometimes refer to a function which is negative with respect to its input variables simply as an <u>incompletely specified negative function</u>.

<u>Theorem 2.3</u> - An incompletely specified function f of n variables is negative with respect to these n variables if and only if for each pair of specified input vectors A, B ε V_n such that f(A)=0 and f(B)=1, there exists a subscript i ($1 \leq i \leq n$) such that a_i=1 and b_i=0.

<u>Proof</u> - First let us prove the "if" part of the theorem. Let h denote a completely specified function obtained by setting the unspecified values of h according to the following rules.

(1) h(A)=0 for every A such that f(A)=0;

(2) h(A)=0 for every A such that f(A)=* and there exists an B ε V_n satisfying B < A and f(B)=0;

(3) h(A)=1 for every A ε V_n which does not satisfy any condition for rule (1) or (2).

Obviously, h is a completion of f, i.e., h(A)=f(A) for every A such that f(A)≠*. We shall prove that h is a negative function. Suppose h is not a negative function. According to Definition 2.1 there must be an inverse edge $\overrightarrow{BA}$ in C_n(h). There are the following four cases:

Case 1 - $f(B)=1$ and $f(A)=0$

Since $\vec{BA}$ is a directed edge in C_n, $b_i \geq a_i$ holds, contradicting the assumption that there exists an i such that $a_i > b_i$.

Case 2 - $f(B)=*$ and $f(A)=0$

According to rule (2) above, $h(B)=0$ must hold, contradicting that $\vec{BA}$ is an inverse edge in $c_n(h)$.

Case 3 - $f(B)=1$ and $f(A)=*$

Since $\vec{BA}$ is an inverse edge in $C_n(h)$, $h(A)=0$ holds. But $h(A)=0$ must have been assigned according to rule (2) above. This means that there exists a $D \varepsilon V_n$ such that $f(D)=0$ and $D < A$. Now let us consider the two vectors B and D. We have $f(B)=1$, $f(D)=0$, and $B > A > D$. Consequently, we have $b_i \geq d_i$ for every i, contradicting the assumption $b_i < d_i$ for some i.

Case 4 - $f(B)=*$ and $f(A)=*$

Since $\vec{BA}$ is an inverse edge in $C_n(h)$, $h(A)=0$ holds. $h(A)=0$ must have been assigned according to rule (2) above. This means that there exists a $D \varepsilon V_n$ such that $f(D)=0$ and $D < A$. This in turn means $f(B)=*$ and $B > A > D$. Then according to rule (2), $h(B)$ must have been assigned the value 0. But this contradicts the assumption that $\vec{BA}$ is an inverse edge in $C_n(h)$.

The above four cases exhaust all possibilities, and consequently the "if" part of the theorem is proved.

The "only if" part of this theorem is obvious from Theorem 2.1 since any negative completion h of f must satisfy $h(A)=f(A)$ for every A such that $f(A)\neq*$.

Q.E.D.

The proof of the above theorem also gives an algorithm to obtain one negative completion from a given incompletely specified negative function. It should be noted, however, that there are generally negative completions other than the h in the above proof for a given function f. The following corollaries are sometimes more convenient in judging whether or not an incompletely specified function is negative.

Corollary 2.2 - A function f of n variables is negative with respect to these n variables if and only if for every vector $A \varepsilon C_n$ such that f(A)=0, we have f(B)=0 or * for every vector $B \varepsilon C_n$ such that $B > A$.

Corollary 2.3 - A function f of n variables is negative with respect to these n variables if and only if for every vector $A \varepsilon C_n$ such that f(A)=1, we have f(B)=1 or * for every vector $B \varepsilon C_n$ such that $B < A$.

In the above we have presented basic properties of negative functions. Next let us consider MOS cells in connection with negative functions.

Fig. 2.5 shows a typical MOS cell which consists of a load FET and several driver FETs. Since the load FET is always conductive, the output terminal is permanently connected to the voltage supply through the load resistance which the load FET constitutes. Each driver FET operates like a relay contact: it is conductive when its input is a logic "1" (represented by a voltage close to the supply voltage), and it is non-conductive when its input is a logic "0"

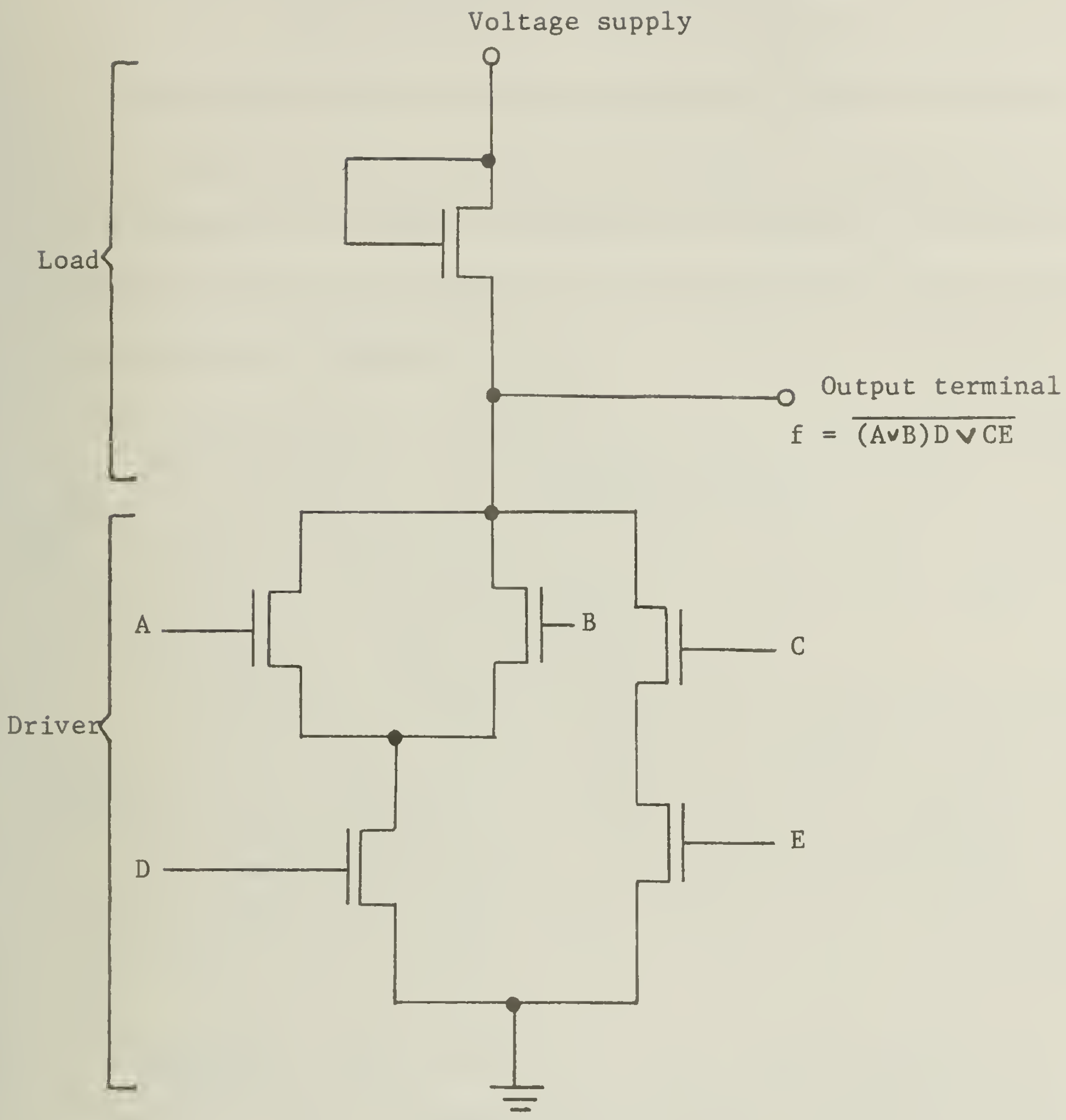


Fig. 2.5 A typical MOS cell.

(represented by a voltage close to the ground voltage). If there is no conductive path between the output terminal and ground, the output will represent a logic "1" since its voltage is almost identical to that of voltage supply. However, if there is a conductive path between the output terminal and ground, the output will represent a logic "0" because its voltage is close to ground due to the relatively high resistance in the load FET. Therefore, the MOS cell shown in Fig. 2.5 realizes the function $\overline{(A \vee B)D \vee CE}$. In general, the driver itself can realize any <u>positive function</u> (i.e., any function that can be written in a disjunctive form with non-complemented variables). The output, however, is always the complement (negation) of the function realized by the driver alone. Thus, theoretically, any negative function can be realized by a single MOS cell. (By Theorem 2.2, a negative function is a function that has a disjunctive form consisting of only complemented variables which, by the duality theorem, is equivalent to a function that is the complement of a positive function.)

In the following discussion, we will use the term "a <u>network with negative gates</u>" or "a <u>negative gate network</u>" to mean a network consisting of only negative gates. Each of such gates realizes a function which is negative with respect to its inputs, and we are not concerned with the structure of each gate. This will be distinguished from "a <u>network with MOS cells</u>" or "an <u>MOS network</u>" which are negative gate networks with each negative gate realized by an MOS cell.

3. ALGORITHMS FOR DESIGNING NETWORKS WITH A MINIMUM NUMBER OF NEGATIVE GATES

This section will discuss two algorithms for the synthesis of networks with a minimum number of negative gates which are basic algorithms for the design procedure of irredundant MOS networks to be introduced in Section 6. Only feed-forward networks, i.e., loop-free networks will be considered.

Definition 3.1 - A minimum negative gate network for a switching function f is a feed-forward network for f that consists of R_f negative gates, where R_f is the minimum number of negative gates required for the realization of f with only negative gates.

Our problem is to design a network for a given function using a minimum number of negative gates alone. K. Nakamura, N. Tokura, and T. Kasami solved this problem by considering the relationship between a general feed-forward negative gate network and the labeled n-cube with respect to the functions realized by the gates in this network. Fig. 3.1 shows a generalized form of a feed-forward network with R negative gates, where $x_1,\ldots,x_n$ denote the n input variables, g_i denotes the i-th gate from the left for $i=1,\ldots,R$ with g_R being the output gate. Let $u_i(x_1,\ldots,x_n)$ denote the function realized by gate g_i with respect to the input variables $x_1,\ldots,x_n$. Since every gate is a negative gate, u_i is negative with respect to the inputs $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ of g_i. In other words, $u_i(x_1,\ldots,x_n,$ $u_1,\ldots,u_{i-1})$ is an incompletely specified negative function of

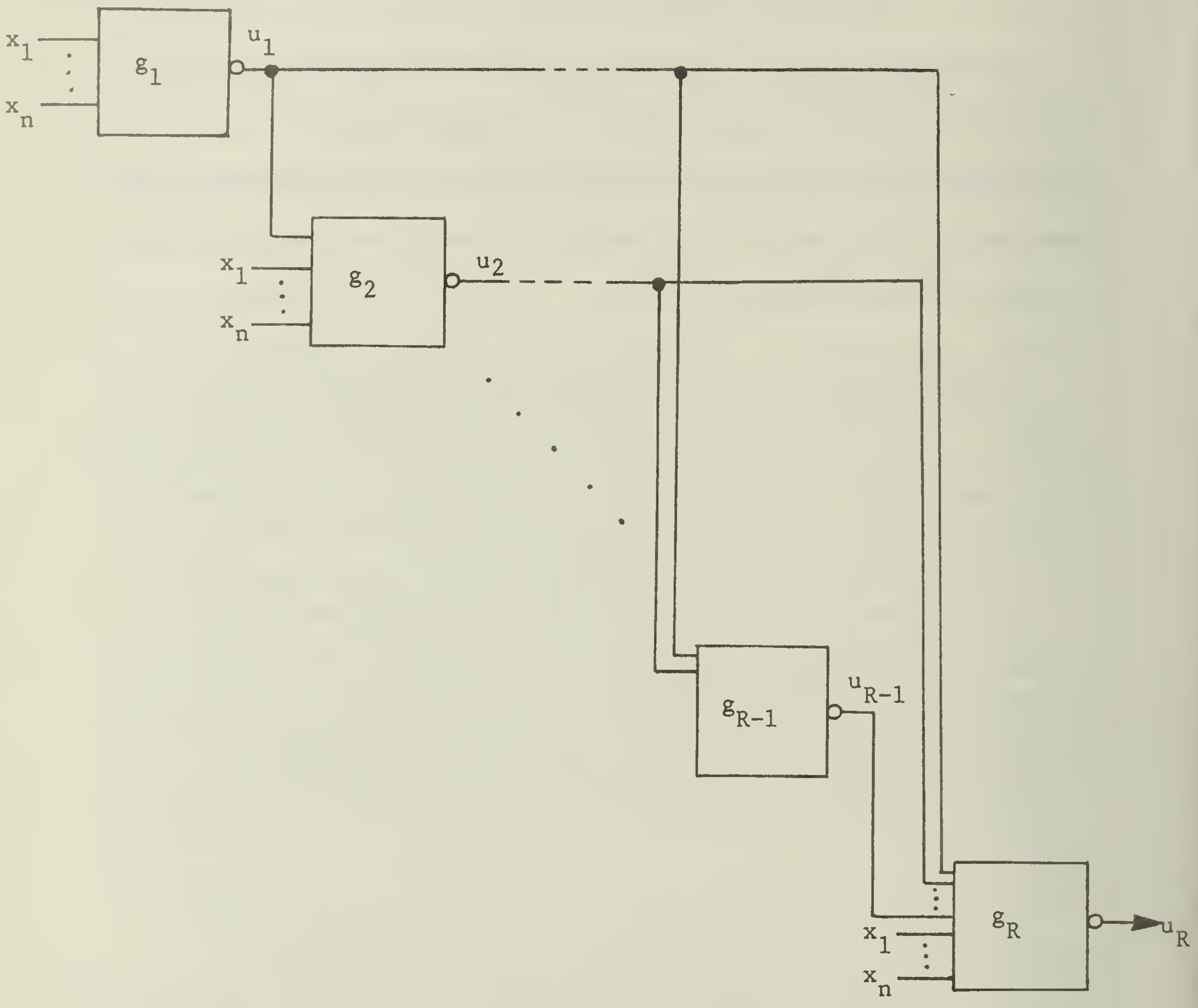


Fig. 3.1 Generalized form of a feed-forward network consisting of R negative gates.

n+i-1 variables. This network is generalized form of a feed-forward network with R negative gates, since every feed-forward network of R negative gates can be expressed in this form with some (or none of the) connections deleted. The missing connections in a particular network do not affect the generality of the generalized form since each function $u_i(x_1,\dots,x_n, u_1,\dots,u_{i-1})$ may contain dummy variables corresponding to the missing connections to gate g_i. For example, if, in a particular network, gate g_1 is not connected to gate g_2, then in the generalized form $u_2(x_1,\dots,x_n,u_1)$ will be a function of $x_1,\dots,x_n$ while u_1 is a dummy variable.

Since this generalized form of Fig. 3.1 represents all possible configurations of networks consisting of R negative gates, the problem of the synthesis of networks with a minimum number of gates for a given function f becomes the problem of finding a sequence of functions $u_1,u_2,\dots,u_R$ such that (1) u_1 is a negative function of $x_1,\dots,x_n$; (2) u_i is negative with respect to $x_1,\dots,x_n$, $u_1,\dots,u_{i-1}$ for $i=2,\dots,R$; (3) $u_R = f$; and (4) R is minimized, i.e., $R=R_f$. For convenience, let us define the following.

<u>Definition 3.2</u> - A <u>negative function sequence of length R for function f</u>, denoted by NFS(R,f) is an ordered set of R functions in C_n, $u_1,u_2,\dots,u_R$, such that

(1) u_i is a negative function in C_n;

(2) u_i is negative with respect to $x_1,\dots,x_n$, $u_1,\dots,u_{i-1}$ for $i=2,\dots,R$; and

(3) $u_R=f$.

Based on the above definition, our problem of designing a minimum negative gate network for a switching function in C_n can be restated as the problem of finding an NFS(R,f)=($u_1,\ldots,u_R \equiv f$) in C_n such that R is minimized (i.e., $R=R_f$). The following lemma is due to [NTK 72] which characterizes an NFS(R,f) in terms of labeled n-cubes.

Theorem 3.1 - A sequence of functions $u_1,u_2,\ldots,u_R$ in C_n is an NFS(R,u_R) if and only if the labeled n-cube with respect to $u_1,\ldots,u_R$, i.e., $C_n(u_1,\ldots,u_R)$, has no inverse edge.

Proof - First let us prove by mathematical induction the "if" part of the theorem. Suppose $C_n(u_1,\ldots,u_R)$ has no inverse edge. Then for every directed edge $\vec{AB} \in C_n$, $u_1(A) \leq u_1(B)$ must hold. Therefore, u_1 is a negative function in C_n, i.e., u_1 is an NFS($1,u_1$). Now suppose $\{u_1,\ldots,u_i\}$ is an NFS(i,u_i). Then we will prove that $\{u_1,\ldots,u_i,u_{i+1}\}$ is an NFS($i+1,u_{i+1}$). In other words, we have to prove u_{i+1} is negative with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_i$. Since u_{i+1} is an incompletely specified function with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_i$, we have to show that for every pair of vectors $(a_1,\ldots,a_n, u_1(A),\ldots,u_i(A))$ and $(b_1,\ldots,b_n, u_1(B),\ldots,u_i(B))$ such that $u_{i+1}(a_1,\ldots,a_n, u_1(A),\ldots,u_i(A))=1$ and $u_{i+1}(b_1,\ldots,b_n, u_1(B),\ldots,u_i(B))=0$, $(a_1,\ldots,a_n, u_1(A),\ldots,u_i(A)) \nless (b_1,\ldots,b_n, u_1(B),\ldots,u_i(B))$ must hold by Theorem 2.3. If A < B or A and B are incomparable, the above condition is obviously satisfied. A=B need not be considered because $(a_1,\ldots,a_n, u_1(A),\ldots,u_i(A))$ and $(b_1,\ldots,b_n, u_1(B),\ldots,u_i(B))$ become the same vector. Therefore we need only consider the case of A > B. Since $C_n(u_1,\ldots,u_R)$, and accordingly $C_n(u_1,\ldots,u_i)$, has no inverse edge, $\ell(B;\ u_1(B),\ldots,u_i(B)) \geq \ell(A;\ u_1(A),\ldots,u_i(A))$ must hold. In other words, only the following three cases can occur:

(1) $(u_1(A),\ldots,u_i(A))$ is incomparable with $(u_1(B),\ldots,u_i(B))$;

(2) $(u_1(A),\ldots,u_i(A)) < (u_1(B),\ldots,u_i(B))$;

(3) $(u_1(A),\ldots,u_i(A)) = (u_1(B),\ldots,u_i(B))$.

For the first two cases, $(a_1,\ldots,a_n,\ u_1(A),\ldots,u_i(A))$ becomes incomparable with $(b_1,\ldots,b_n,\ u_1(B),\ldots,u_i(B))$. For the third case, $u_{i+1}(A)=1$ and $u_{i+1}(B)=0$ can never occur because otherwise $\ell(A;\ u_1,\ldots,u_{i+1}) > \ell(B;\ u_1,\ldots,u_{i+1})$ holds which contradicts the assumption that $C_n(u_1,\ldots,u_R)$ and accordingly $C_n(u_1,\ldots,u_{i+1})$ has no inverse edge. Therefore, when $u_{i+1}(a_1,\ldots,a_n,\ u_1(A),\ldots,u_i(A))=1$ and $u_{i+1}(b_1,\ldots,b_n,\ u_1(B),\ldots,u_i(B))=0$ holds, $(a_1,\ldots,a_n,\ u_1(A),\ldots,u_i(A)) \nleq (b_1,\ldots,b_n,\ u_1(B),\ldots,u_i(B))$ holds. Then by Theorem 2.3, u_{i+1} is negative with respect to $x_1,\ldots,x_n,\ u_1,\ldots,u_i$. Therefore, $\{u_1,\ldots,u_{i+1}\}$ is an NFS$(i+1,\ u_{i+1})$, and consequently $\{u_1,\ldots,u_R\}$ by induction is an NFS $(R,\ u_R)$.

Next, let us prove the "only if" part of this theorem. By definition, u_1 is a negative function with respect to $x_1,\ldots,x_n$, and therefore $C_n(u_1)$ has no inverse edge. Assume $C_n(u_1,\ldots,u_i)$ has no inverse edge. We will prove that $C_n(u_1,\ldots,u_{i+1})$ has no inverse edge if $\{u_1,\ldots,u_R\}$ is an NFS(R,u_R). For each directed edge $\vec{AB} \in C_n$, one of the following two relations must hold:

(1) $\ell(A;\ u_1(A),\ldots,u_i(A)) < \ell(B;\ u_1(B),\ldots,u_i(B))$;

(2) $\ell(A;\ u_1(A),\ldots,u_i(A)) = \ell(B;\ u_1(B),\ldots,u_i(B))$.

In case (1), $\ell(A;\ u_1(A),\ldots,u_i(A),u_{i+1}(A)) < \ell(B;\ u_1(B),\ldots,u_i(B),$ $u_{i+1}(B))$ holds regardless the values of $u_{i+1}(A)$ and $u_{i+1}(B)$. In case (2), $u_1(A) = u_1(B),\ldots,u_i(A) = u_i(B)$ results. Since u_{i+1} is negative with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_i$, we must have $u_{i+1}(A) \leq u_{i+1}(B)$ because if $u_{i+1}(A) > u_{i+1}(B)$ holds, Theorem 2.3 will not be satisfied. Therefore, $\ell(A;\ u_1(A),\ldots,u_i(A),u_{i+1}(A)) \leq \ell(B;\ u_1(B),\ldots,u_i(B),$ $u_{i+1}(B))$ holds. Thus $C_n(u_1,\ldots,u_{i+1})$ has no inverse edge. By induction $C_n(u_1,\ldots,u_R)$ has no inverse edge.

Q.E.D.

Based on Theorem 3.1, an algorithm for the synthesis of a network realizing a given function f with a minimum number of negative gates has been derived by [NTK 72]. The basic concept of this algorithm is to find a labeled n-cube $C_n(u_1,\ldots,u_R)$ such that (1) $u_R=f$; (2) $C_n(u_1,\ldots,u_R)$ has no inverse edge; and (3) the label for vertex A, $\ell(A;\ u_1,\ldots,u_R)$, is minimized under the above two conditions for each $A \in C_n$, and therefore R, the bit length of binary numbers $\ell(A;\ u_1,\ldots,$ $u_R)$, is minimized. This algorithm will be referred to as the <u>algorithm based on minimum labeling</u> or <u>algorithm MNL</u>. The symbol $L^f_{mn}(X)$ will be used to denote the label assigned to vertex X by this algorithm for function f.

<u>Algorithm 3.1</u> - Algorithm based on minimum labeling (<u>MNL</u>).

<u>Step 1</u> - Assign as $L^f_{mn}(I)$ the value of f(I).

<u>Step 2</u> - When $L^f_{mn}(A)$ is assigned to every vertex A of weight $w(1 \leq w \leq n)$ in C_n, assign as $L^f_{mn}(B)$ to each vertex B of weight w-1 the smallest binary integer satisfying the following two conditions:

(a) The least significant bit of $L^f_{mn}(B)$ is $f(B)$;

(b) $L^f_{mn}(B) \geq L^f_{mn}(A)$ for every directed edge $\vec{AB}$ terminating at B.

<u>Step 3</u> - Repeat Step 2 until $L^f_{mn}(0)$ is assigned.

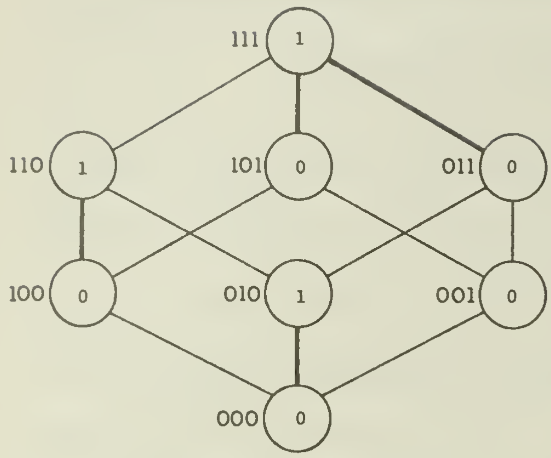
<u>Step 4</u> - The number of bits in $L^f_{mn}(0)$ is R_f, the minimum number of negative gates required to realize f, and the i-th significant bit of $L^f_{mn}(A)$ is $u_i(A)$ for each $A \in C_n$ and for $i = 1,\ldots R_f$.

The validity of this algorithm is obvious from Theorem 3.1. The algorithm obtains an $NFS(R_f,f) = (u_1,\ldots,u_{R_f} \equiv f)$ where R_f is the number of bits in $L^f_{mn}(0)$. From the definition of $NFS(R_f,f)$, u_i for $i = 1,\ldots,$ R_f can be realized by a negative gate network of the form of Fig. 3.1. Since the minimum possible $L^f_{mn}(A)$ to guarantee no inverse edge in the labeled n-cube is chosen at each vertex $A \in C_n$, R_f is the minimum number of negative gates required to realize f by a feed-forward configuration of negative gates.

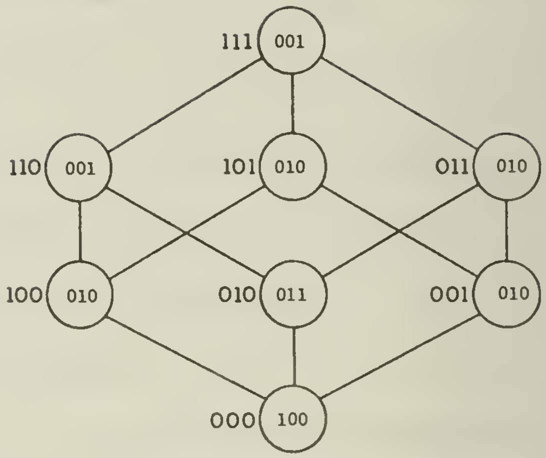
<u>Example 3.1</u> - Design a minimum negative gate network for $f = x_1x_2 \vee \bar{x}_1x_2\bar{x}_3$ in C_3.

The true vertices of $C_3(f)$ are (111), (110) and (010) as shown in Fig. 3.2(a). According to Algorithm 3.1 (MNL), the labeled 3-cube $C_3(u_1,u_2,f)$ is obtained as shown in Fig. 3.2(b). (u_1,u_2,f) as an NFS(3,f) can then be realized by a network with three negative gates in the generalized form as shown in Fig. 3.2(c).

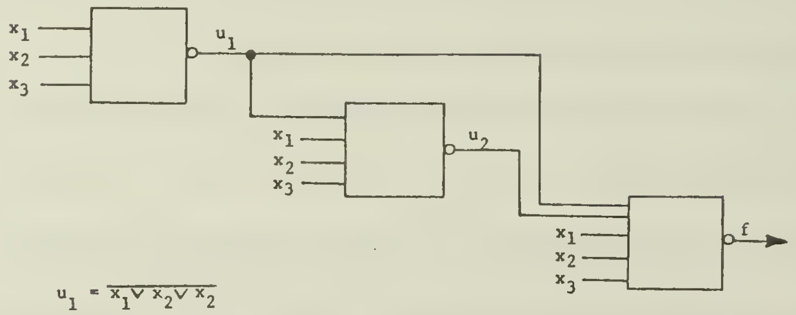
<u>Corollary 3.1</u> - Let $NFS(R_f,f) = (u_1,\ldots,u_{R_f})$ be a minimum negative function sequence for f, and $C_n(u_1,\ldots,u_{R_f})$ be the corresponding labeled n-cube. Then $\ell(A; u_1,\ldots,u_{R_f}) \geq L^f_{mn}(A)$ must hold for every vertex $A \in C_n$.



(a) $C_3(f)$ for $f = x_1x_2 \vee x_2x_3$ (bold lines denote inverse edges).



(b) $C_3(u_1,u_2,f)$ based on minimal labeling.



$u_1 = \overline{x_1 \vee x_2 \vee x_2}$

$u_2 = (\bar{x}_1 \vee \bar{x}_2)\ (x_1 \vee x_2 \vee x_3) = \overline{u_1 \vee x_1x_2}$

$u_3 = x_1x_2 \vee x_2\bar{x}_3 = \overline{u_1 \vee u_2(x_3 \vee x_1)}$

(c) Minimum negative gate network for f based on minimum labeling.

Fig. 3.2 Example 3.1.

Next, let us investigate the label attached to each vertex by MNL. Theorem 3.2, to be introduced following Definitions 3.3, 3.4, and 3.5, characterizes the label attached to each vertex by algorithm MNL.

Definition 3.3 - A directed path from a vertex A_1 to another vertex A_p satisfying $A_1 > A_p$ in an n-cube C_n is a sequence of directed edges connecting the two vertices, $\overrightarrow{A_1A_2}$, $\overrightarrow{A_2A_3}, \ldots,$ $\overrightarrow{A_{p-1}A_p}$.

It is obvious that there may be more than one directed path between two vertices in C_n.

Definition 3.4 - In a labeled n-cube $C_n(f)$ for a function f, the number of inversions in a directed path between two vertices is the number of inverse edges included in this path. The inversion degree of a vertex A with respect to vertex I in $C_n(f)$ is the maximum number of inversions over all directed paths from vertex I to vertex A, and is denoted by $D_I^f(A)$.

Definition 3.5 - The number of inversions of a function f is the maximum number of inversions over all directed paths from vertex I to vertex 0 in $C_n(f)$, i.e., $D_I^f(0)$.

Example 3.2 - In Example 3.1, $D_I^f(111) = D_I^f(110) = 0$, $D_I^f(100) = D_I^f(101) = D_I^f(011) = D_I^f(010) = D_I^f(001) = 1$, and $D_I^f(000) = 2$ (Fig. 3.2(a)).

The number of inversions of a function determines the number of negative gates required to realize this function. This relation is stated in Corollary 3.2 which is an immediate consequence of Theorem 3.2.

Theorem 3.2 - $L^f_{mn}(X)$, the label attached to vertex X by Algorithm 3.1 (MNL) for a function f has the value $L^f_{mn}(X) = 2\ D^f_I(X) + f(X)$ for every $X \varepsilon C_n$.

Proof - According to Theorem 3.1 and also $D^f_I(I) = 0$ by definition, $L^f_{mn}(I) = f(I) = 2\ D^f_I(I) + f(I)$. Therefore, the theorem holds for the vertex of weight n. Next, assume $L(X) = 2\ D^f_I(X) + f(X)$ holds for every vertex of weight w or greater, and we will prove that it holds for every vertex of weight w-1. Let us consider a vertex B of weight w-1 and select a vertex A of weight w such that

$$L^f_{mn}(A) = \max \{L^f_{mn}(X) \mid \vec{XB} \varepsilon C_n\} \qquad (3.1)$$

Because $L^f_{mn}(X) = 2\ D^f_I(X) + f(X)$ holds for every X of weight w or greater,

$$L^f_{mn}(A) = 2\ D^f_I(A) + f(A) \qquad (3.2)$$

holds. Since $L^f_{mn}(A)$ is chosen as the maximum of $L^f_{mn}(X)$ by (3.1) and f(A) is a constant in (3.2),

$$D^f_I(A) = \max \{D^f_I(X) \mid \vec{XB} \varepsilon C_n\} \qquad (3.3)$$

must hold. The following cases may occur:

(1) $f(A)=0$, i.e., $L^f_{mn}(A) = 2\ D^f_I(A)$. According to Step 2 of algorithm MNL, $L^f_{mn}(B)$ will be assigned the value satisfying $L^f_{mn}(B) = L^f_{mn}(A) + f(B) = 2\ D^f_I(A) + f(B)$. Since $\vec{AB}$ does not constitute

an inverse edge in $C_n(f)$ and (3.3) holds (the case when $D_I^f(C) = D_I^f(A) = \max \{D_I^f(X) \mid \vec{XB} \in C_n\}$ holds for another vertex C as well as for A, f(C)=1 can not occur since otherwise $L_{mn}^f(C) = 2\,D_I^f(C) + f(C) > 2\,D_I^f(A) + f(A) = L_{mn}^f(A)$ hold, contradicting (3.1)), we have $D_I^f(B) = D_I^f(A)$. Therefore, $L_{mn}^f(B) = 2\,D_I^f(B) + f(B)$ holds.

(2) f(A)=1, f(B)=1. None of $\vec{XB}$ is an inverse edge in C_n, so we have $D_I^f(B) = D_I^f(A)$. According to algorithm MNL, $L_{mn}^f(B) = L_{mn}^f(A) = 2\,D_I^f(B) + f(B)$ must hold.

(3) f(A)=1, f(B)=0. $\vec{AB}$ is an inverse edge in C_n, so $D_I^f(B) = D_I^f(A) + 1$. According to algorithm MNL, $L_{mn}^f(B) = L_{mn}^f(A) + 1 = 2\,D_I^f(A) + f(A) + 1 = 2\,D_I^f(B) + f(B)$ must hold.

Consequently, $L_{mn}^f(B) = 2\,D_I^f(B) + f(B)$ holds for every vertex B of weight w-1. By induction, $L_{mn}^f(X) = 2\,D_I^f(X) + f(X)$ holds for every $X \in C_n$.

Q.E.D.

Corollary 3.2 - R_f, the minimum number of negative gates required to realize function f is given by

$$R_f = \lceil \log_2(D_I^f(0) + 1) \rceil + 1$$

where $\lceil r \rceil$ denotes the smallest integer not smaller than r.

Proof - By Lemma 3.2, $L_{mn}^f(0) = 2\,D_I^f(0) + f(0)$ which requires R_f bits to be represented in binary. This means that R_f is the smallest integer satisfying

$$2^{R_f} - 1 \geq 2\, D_I^f(0) + f(0) \qquad (3.4)$$

Therefore,

$$R_f = \lceil \log_2 (2\, D_I^f(0) + f(0) + 1) \rceil = \lceil \log_2 (D_I^f(0) + 1) \rceil + 1$$

Q.E.D.

From equation (3.4), it is obvious that

$$D_I^f(0) \leq 2^{R_f - 1} - 1 \qquad (3.5)$$

is always satisfied. This relation will be used later.

The above relations are consistent with the results obtained by A. A. Markov [Mar 58] and independently by D. E. Muller [Mul 58] as discussed in [NTK 72].

Next let us consider the labeled n-cube $C_n(u_1, \ldots, u_{R_f - 1}, f)$ obtained by algorithm MNL for a function f. We can partition the entire set of vertices in C_n into disjoint subsets such that two vertices A and B belong to the same subset if and only if $L_{mn}^f(A) = L_{mn}^f(B)$. It is obvious that if A and B are in the same subset, f(A) = f(B) must hold, i.e., A and B must be both true or false vectors. Each of the above subsets is called a _cluster based on MNL_.

In general, given any labeled n-cube $C_n(u_1, \ldots, u_{R_f - 1}, f)$ that has no inverse edges, we can partition the entire set of vertices into disjoint subsets in the same manner as above. Each of these subsets is simply called a _cluster_, in contrast to "a cluster based on MNL" above. A _true cluster_ is a cluster which consists of true vectors, whereas a _false cluster_ is a cluster which consists of false vectors. An n-cube where all vertices are partitioned into clusters is called a <u>clustered n-cube</u>. The concept of clustered n-cube is a generalization of the concept of the stratified structure of a switching function defined in [Liu 72]. We will show later that the stratified structure of a switching function is essentially the clustered n-cube based on Algorithm 3.2 (MXL) to be introduced later.

In general, the clustered n-cube based on algorithm MNL is different from the clustered n-cube based on algorithm MXL, and accordingly different from the stratified structure of [Liu 72]. Nevertheless, most of the properties possessed by the stratified structure of a switching function are also possessed by the clustered n-cube based on MNL. This will be discussed in more detail later.

The negative gate network for a function f produced by algorithm MNL is usually not the only minimum negative gate network for f. If $D_I^f(0) \neq 2^{R_f-1} - 1$, we can relabel the vertices in C_n according to the following.

<u>Relabeling of a clustered n-cube based on MNL</u>

(1) Arrange the clusters based on MNL in such an order that a cluster with a smaller label proceeds a cluster with a greater label (assigned by algorithm MNL).

(2) Assign to every vertex in each cluster the same label of R_f bits such that: (a) the last bit of the label is "1" or "0" depending on whether this cluster is a true or false cluster, respectively; and (b) the label assigned to each cluster must be greater than those assigned to clusters preceding it.

Through relabeling, we can obtain a different minimum negative gate network. This way of relabeling is similar to the procedure given in [Liu 72] where a realizable stratified truth table (corresponding to a labeled n-cube without inverse edge) can be obtained by assigning each cluster (based on MXL to be introduced) a unique label such that the resulting labeled n-cube has no inverse edge. Example 3.3 shows how one can obtain different minimum negative gate networks from the clustered n-cube based on MNL by relabeling.

Example 3.3 - For the function in Example 3.1, there are four clusters based on MNL (see Fig. 3.2(b)).

1. True cluster: (111), (110).
2. False cluster: (101), (011), (100), (001).
3. True cluster: (010).
4. False cluster: (000).

Since $D_I^f(0) = 2$, at least three negative gates are required to realize f according to Corollary 3.2. Table 3.1 shows five different

Clusters based on MNL	Vertices	Different NFS(3,f)'s														
		# 1			# 2			# 3			# 4			# 5		
		u_1	u_2	f	u_1	u_2	f	u_1	u_2	f	u_1	u_2	f	u_1	u_2	f
1	(111),(110)	0	0	1	0	0	1	0	0	1	0	0	1	0	1	1
2	(101),(011),(100),(001)	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0
3	(010)	0	1	1	0	1	1	1	0	1	1	0	1	1	0	1
4	(000)	1	0	0	1	1	0	1	1	0	1	1	0	1	1	0

Table 3.1 Possible NFS(3,f)'s obtained from clusters based on MNL for function $f = x_1x_2 \vee x_2\bar{x}_3$.

labelings obtained by relabeling the clustered n-cube based on MNL. All five labelings result in different NFS(3,f) ($NFS(3,f)_2$ and $NFS(3,f)_4$ are indistinguishable as will be discussed in Section 4). It should be noted that $NFS(3,f)_1$ in Table 3.1 is the one obtained by algorithm MNL.

As we can see from this simple example, we may have different ways in attaching labels for a given clustered n-cube, and yet the resulting labeled n-cubes have no inverse edges. If $D_I^f(0) = 2^{R_f - 1} - 1$, we have only one way to label these clusters with R_f bits for each label. Even if $D_I^f(0) = 2^{R_f - 1} - 1$, however, the way of clustering is usually not unique, and therefore the labeled n-cube (the $NFS(R_f,f)$) is usually not unique. One way to generate a different clustering is to use Algorithm 3.2 to be introduced in the following.

Algorithm 3.2 is based on the idea of attaching to each vertex in an n-cube a maximum possible label consisting of a minimum number of bits such that the resulting n-cube has no inverse edge. From this labeled n-cube, an $NFS(R_f,f)$ is obtained where $R_f = \lceil \log_2 (D_I^f(0) + 1) \rceil + 1$. The algorithm is called the <u>algorithm based on maximum labeling with a minimal number of bits</u> and will be referred to as <u>algorithm MXL</u>.

<u>Algorithm 3.2</u> - Algorithm based on maximum labeling with a minimum number of bits (<u>MXL</u>).

Let $L^f_{mx}(A)$ be the label attached to each vertex $A \in C_n$ by this algorithm for a given function f. Let $R_f = \lceil \log_2 (D^f_I(0) + 1) \rceil + 1$

<u>Step 1</u> - Assign as $L^f_{mx}(0)$ the value of $2^{R_f} - 2 + f(0)$.

<u>Step 2</u> - When $L^f_{mx}(A)$ is assigned to every vertex A of weight w ($0 \leq w \leq n-1$) in C_n, assign as $L^f_{mx}(B)$ to each vertex B of weight w+1 the largest binary integer satisfying the following two conditions:

(a) The least significant bit of $L^f_{mx}(B)$ is f(B);

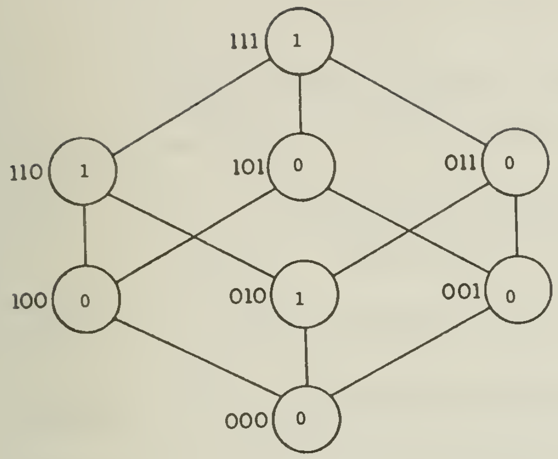
(b) $L^f_{mx}(B) \leq L^f_{mx}(A)$ for every directed edge $\vec{BA}$ originating from B.

<u>Step 3</u> - Repeat Step 2 until $L^f_{mx}(I)$ is assigned.

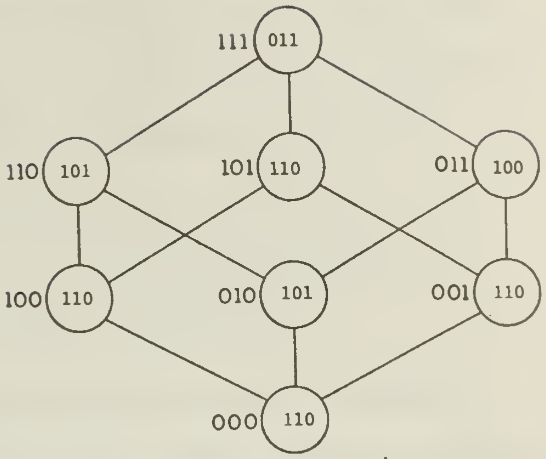
<u>Step 4</u> - Let the i-th significant bit of $L^f_{mx}(A)$ be $u_i(A)$ for every $A \in C_n$. Then the resulting $(u_1,\ldots,u_{R_f})$ where $R_f = \lceil \log_2(D^f_I(0) + 1) \rceil + 1$ is an NFS(R_f,f) for function f.

<u>Example 3.4</u> - Consider the function discussed in Example 3.1, $f = x_1x_2 \vee x_2\bar{x}_3$. Fig. 3.3(a) shows $C_3(f)$. As seen in Example 3.1, $D^f_I(0) = 2$ and three negative gates are required. According to MXL, $2^3 - 2 + f(0) = 110$ is assigned to vertex 0 as $L^f_{mx}(0)$, and the labeled 3-cube is completed according to algorithm MXL as shown in Fig. 3.3(b). Fig. 3.3(c) shows the negative gate network obtained in the general form. The actual network can be obtained by deleting certain connections from this network, as will be discussed in a later section.

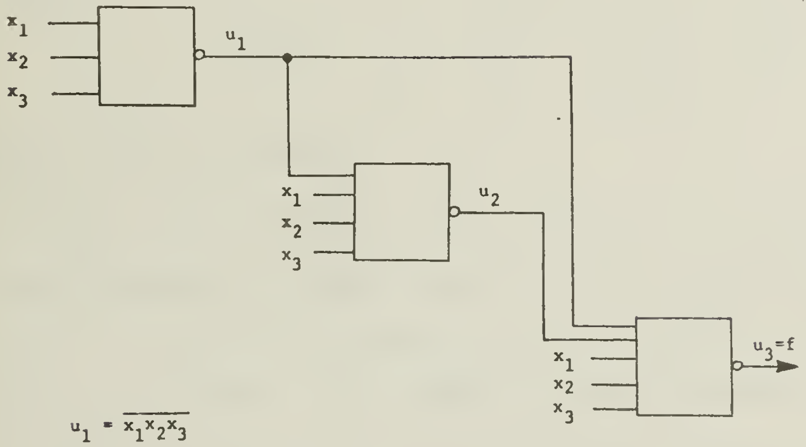
The validity of this algorithm is obvious from Theorem 3.1 and the fact that the number of inversions of function f satisfy $D^f_I(0) \leq 2^{R_f-1} -1$ by (3.5). The following discussion gives a formal proof.



(a) $C_3(f)$ for $f = x_1x_2 \vee x_2\bar{x}_3$.



(b) $C_3(u_1,u_2,f)$ based on MXL.



$u_1 = \overline{x_1x_2x_3}$

$u_2 = \bar{x}_2 \vee x_1x_2x_3 = \overline{x_2\bar{x}_1}$

$u_3 = x_1x_2 \vee x_2\bar{x}_3 = \overline{u_1(u_2 \vee x_2x_3)}$

(c) A minimum negative gate network for f based on MXL.

<u>Fig. 3.3</u> Example 3.4.

Definition 3.6 - The inversion degree of a vertex A with respect to vertex 0 in $C_n(f)$ is the maximum number of inversions over all directed paths from vertex A to vertex 0 and is denoted by $D_0^f(A)$.

From this definition and the definition of the number of inversions of a function, the following equation is obvious:

$$D_0^f(I) = D_I^f(0)$$

Similar to Theorem 3.2, we have Theorem 3.3 which characterizes the relationship between $L_{mx}^f(X)$ and $D_0^f(X)$ for $X \varepsilon C_n$.

Theorem 3.3 - $L_{mx}^f(X)$, the label attached to a vertex $X \varepsilon C_n$ by algorithm MXL for function f, has the value

$$L_{mx}^f(X) = 2^{R_f} - 2\ (D_0^f(X) + 1) + f(X).$$

Proof - By definition, $D_0^f(0) = 0$ holds. According to Algorithm 3.2, $L_{mx}^f(0) = 2^{R_f} - 2 + f(0) = 2^{R_f} - 2\ (D_0^f(0) + 1) + f(0)$ holds, i.e., the theorem holds for the vertex of weight 0. Assume $L_{mx}^f(X) = 2^{R_f} - 2(D_0^f(X) + 1) + f(X)$ holds for each vertex X of weight w or less. Consider an arbitrary vertex A of weight w+1. Select a vertex B of weight w such that

$$L_{mx}^f(B) = \min \{L_{mx}^f(X) \mid \vec{AX} \varepsilon C_n\} \tag{3.6}$$

Because $L_{mx}^f(X) = 2^{R_f} - 2(D_0^f(X) + 1) + f(X)$ holds for every X of weight w or less, we have

$$L_{mx}^f(B) = 2^{R_f} - 2\ (D_0^f(B) + 1) + f(B) \tag{3.7}$$

Since $L^f_{mx}(B)$ is chosen as the minimum by (3.6) and $f(B)$ is constant in (3.7),

$$D^f_0(B) = \min \{D^f_0(X) \mid \vec{AX} \in C_n\} \qquad (3.8)$$

must hold. The following cases may occur:

(1) $f(B)=1$. According to Step 2 of algorithm MXL, $L^f_{mx}(A)$ is assigned a maximum possible value less than or equal to $L^f_{mx}(B) = 2^{R_f} - 2(D^f_0(B) + 1) + f(B)$. Because $\vec{AB}$ does not constitute an inverse edge in $C_n(f)$ and (3.8) holds (the case when $D^f_0(C) = D^f_0(B) = \min \{D^f_0(X) \mid \vec{AX} \in C_n\}$ holds for another vertex C and we have $f(C)=0$ can not occur since otherwise $L^f_{mn}(C) = 2^{R_f} - 2(D^f_0(C) + 1) + f(C) < 2^{R_f} - 2(D^f_0(B) + 1) + f(B) = L^f_{mn}(B)$ hold, contradicting (3.6)), we have $D^f_0(A) = D^f_0(B)$. Therefore $L^f_{mx}(A) = 2^{R_f} - 2(D^f_0(B) + 1) + f(A) = 2^{R_f} - 2(D^f_0(A) + 1) + f(A)$ holds.

(2) $f(B)=0$, $f(A)=0$. None of $\vec{AX}$ is an inverse edge in $C_n(f)$, so $D^f_0(A) = D^f_0(B)$ holds. According to algorithm MXL, $L^f_{mn}(A) = L^f_{mx}(B) = 2^{R_f} - 2(D^f_0(B) + 1) + f(B) = 2^{R_f} - 2(D^f_0(A) + 1) + f(A)$ holds.

(3) $f(B)=0$, $f(A)=1$. $\vec{AB}$ is an inverse edge in $C_n(f)$, so $D^f_0(A) = D^f_0(B) + 1$. According to algorithm MXL, $L^f_{mx}(A) = L^f_{mx}(B) - 1 = 2^{R_f} - 2(D^f_0(B) + 1) - 1 = 2^{R_f} - 2(D^f_0(B) + 2) + 1 = 2^{R_f} - 2(D^f_0(A) + 1) + f(A)$ must hold.

Therefore, $L^f_{mx}(A) = 2^{R_f} - 2(D^f_0(A) + 1) + f(A)$ holds for every A of weight w+1. By induction, $L^f_{mx}(X) = 2^{R_f} - 2(D^f_0(X) + 1) + f(X)$ holds for every $X \varepsilon C_n$.

Q.E.D.

From Theorem 3.3, it is clear that $L^f_{mx}(I) = 2^{R_f} - 2(D^f_0(I) + 1) + f(I) \geq 0$ since $D^f_0(I) = D^f_0(0) \leq 2^{R_f - 1} - 1$ holds by (3.5). This means that when we use algorithm MXL, we can continue labeling up through the vertex I. This proves the validity of algorithm MXL because the resulting labeled n-cube is indeed a labeled n-cube with a minimum number of bits in each label which has no inverse edges.

Having proved the validity of algorithm MXL, now we can present Corollary 3.3 which immediately follows algorithm MXL.

<u>Corollary 3.3</u> - Let $NFS(R_f,f) = (u_1,\ldots,u_{R_f} = f)$ be a minimum negative function sequence for f, and $C_n(u_1,\ldots,u_{R_f})$ be the corresponding labeled n-cube. Then $(A; u_1,\ldots,u_{R_f}) \leq L^f_{mx}(A)$ must hold for every $A \varepsilon C_n$.

Similar to the discussion following algorithm MNL, we can partition the entire set of vertices into clusters such that all vertices in each cluster have the same label based on MXL. These clusters based on MXL constitute what T. K. Liu defined as stratified structure of a switching function [Liu 72].

<u>Definition 3.7</u> - The <u>stratified structure</u> of a function of n variables is a sequence of subsets of input vectors, $(M^f_0, M^f_1,\ldots,M^f_{2r})$ where $r = D^f_0(I)$, defined as:

(i) M^f_{2i+1} for $i = 0,\ldots,r-1$, contains every vector A such that $f(A)=1$ and $D^f_0(A) = i$.

(ii) M^f_{2i} for $i = 0,\ldots,r$, contains every vector A such that $f(A)=0$ and $D^f_0(A) = i$.

The wording of this definition is different from that in [Liu 72], but they define the same concept, the stratified structure of a function. T. K. Liu discussed various properties of the stratified structure of a function which we will not repeat here. However, it should be emphasized that the algorithm based on the maximal labeling attaches to all vertices in each cluster the same label unique to that cluster because of Theorem 3.3.

He also gave several algorithms to design minimum negative gate networks for a given function. All the networks designed by his algorithms are <u>based on the stratified structure of that function</u>. In other words, in the labeled n-cube corresponding to a minimum negative gate network given by algorithms in [Liu 72], all vertices in each M^f_i are assigned the same label unique to that M^f_i. It implies that, in general, the algorithms given in [Liu 72] consider only a subset of all possible NFS(R_f,f)'s for the given function f. Sometimes, these algorithms design minimum negative gate networks whose corresponding MOS networks, regardless how well each cell is designed, may contain redundant FETs and connections. This problem will be discussed in detail in Sections 5 and 6 along with illustrating examples.

Example 3.5 - Consider the function discussed in Examples 3.1, 3.2, 3.3, and 3.4, $f = x_1x_2 \vee x_2\bar{x}_3$. The stratified structure for f is:

$$M_0^f = \{(000), (001), (100), (101)\}$$

$$M_1^f = \{(010), (110)\}$$

$$M_2^f = \{(011)\}$$

$$M_3^f = \{(111)\}$$

All possible NFS(3,f)'s based on this stratified structure for f are listed in Table 3.2.

	Different NFS(3,f)'s														
	1			2			3			4			5		
Cluster	u_1	u_2	f	u_1	u_2	f	u_1	u_2	f	u_1	u_2	f	u_1	u_2	f
M_3^f	0	0	1	0	0	1	0	0	1	0	0	1	0	1	1
M_2^f	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0
M_1^f	0	1	1	0	1	1	1	0	1	1	0	1	1	0	1
M_0^f	1	0	0	1	1	0	1	1	0	1	1	0	1	1	0

Table 3.2 NFS(3,f)'s based on the stratified structure for $f = x_1x_2 \vee x_2\bar{x}_3$.

4. UNIQUENESS OF MINIMUM NEGATIVE GATE NETWORKS

As the examples in Section 3 show, the number of different $NFS(R_f,f)$ for a switching function f is usually more than one. In this section, the problem of when a function has a unique $NFS(R_f,f)$, i.e., a unique minimum negative gate network will be explored.

Definition 4.1 - Let $NFS(R_f,f)_1 = (u_{11},u_{21},\ldots,u_{R_f-1,1},f)$ and $NFS(R_f,f)_2 = (u_{12},u_{22},\ldots,u_{R_f-1,2},f)$ be two minimum negative function sequences for a function f in C_n. $NFS(R_f,f)_1$ and $NFS(R_f,f)_2$ are said to be different if and only if there exist some i $(1 \leq i \leq R_f-1)$ and some vertex $A \varepsilon C_n$ such that $u_{i1}(A) \neq u_{i2}(A)$. $NFS(R_f,f)_1$ and $NFS(R_f,f)_2$ are said to be undistinguishable if and only if $(u_{12},\ldots,u_{R_f-1,2},f)$ is a permutation of $(u_{11},\ldots,u_{R_f-1,1},f)$; otherwise they are said to be distinguishable.

According to this definition, two different $NFS(R_f,f)$'s correspond to the essentially identical network realizations (not speaking of the internal structure of each cell) if they are indistinguishable. But if they are distinguishable, two $NFS(R_f,f)$'s correspond to different negative gate network realizations.

Example 4.1 - In Example 3.3, five NFS(3,f)'s based on the clusters by algorithm MNL for function $f = x_1x_2 \vee x_2\bar{x}_3$ were listed in Table 3.1. From this table, it is clear that $NFS(3,f)_2 =$ $(u_{12} = \overline{x_1 \vee x_2 \vee x_3}, u_{22} = \overline{x_1x_2}, f)$ and $NFS(3,f)_4 = (u_{14} = \overline{x_1x_2},$ $u_{24} = \overline{x_1 \vee x_2 \vee x_3}, f)$ are indistinguishable. All other pairs from the five are distinguishable, i.e., they are representing different minimum negative gate networks.

Definition 4.2 - An NFS(R_f,f) for a switching function f is unique if and only if there exists no other distinguishable NFS(R_f,f).

Lemma 4.1 gives a sufficient condition for a function not to have a unique minimum negative gate network.

Lemma 4.1 - A function f has at least two distinguishable NFS(R_f,f)'s if $D_0^f(I) = D_I^f(0) \neq 2^{R_f-1} - 1$ holds.

Proof - Let $\text{NFS}(R_f,f)_1 = (u_{11},\ldots,u_{R_f-1,1},f)$ and $\text{NFS}(R_f,f)_2 = (u_{12},\ldots,u_{R_f-1,2},f)$ be two NFS(R_f,f)'s generated by algorithms MNL and MXL, respectively. According to MNL, $L_{mn}^f(0) = 2\,D_I^f(0) + f(0)$, and according to MXL, $L_{mx}^f(0) = 2^{R_f} - 2 + f(0)$. Since $D_I^f(0) \neq 2^{R_f-1} - 1$ holds by assumption, $L_{mn}^f(0) = 2\,D_I^f(0) + f(0) \neq 2^{R_f} - 2 + f(0) = L_{mx}^f(0)$ holds. Furthermore, according to MXL, $u_{12}(0) = u_{22}(0) = \ldots = u_{R_f-1,2}(0) = 1$ holds and there exists at least one i ($1 \leq i \leq R_f-1$) such that $u_{i1}(0) = 0$ since $L_{mn}^f(0) \neq L_{mx}^f(0)$. Therefore, $\text{NFS}(R_f,f)_2$ can not be a permutation of $\text{NFS}(R_f,f)_1$, i.e., $\text{NFS}(R_f,f)_1$ and $\text{NFS}(R_f,f)_2$ are distinguishable.

Q.E.D.

According to this lemma, if a function f has a unique NFS(R_f,f), $D_0^f(I) = 2^{R_f-1} - 1$ must hold. The following lemma presents a property of NFS(R_f,f)'s, when $D_0^f(I) = 2^{R_f-1} - 1$ is satisfied.

Lemma 4.2 - For a switching function f, if $D_0^f(I) \equiv D_I^f(0) = 2^{R_f-1} - 1$ holds, then every pair of different NFS(R_f,f)'s (if they exist) are distinguishable.

Proof - Since $D_0^f(I) = D_I^f(0) = 2^{R_f-1} - 1$ holds, there must exist in $C_n(f)$ a directed path from vertex I to vertex 0 such that the number of inversions (i.e., number of inverse edges) along this path equals $2^{R_f-1} - 1$. Let $A_1 > A_2 > \ldots > A_{2D_I^f(0)}$ be $2D_I^f(0)$ vertices along such a path where $f(A_1)=1$, $f(A_2)=0, \ldots,$ $f(A_{2i-1})=1$, $f(A_{2i})=0, \ldots,$ $f(A_{2D_I^f(0)})=0$ hold. It is obvious that there is only one way to label these vertices with R_f bits such that the resulting labeled n-cube has no inverse edge; that is, attaching to each vertex A_i the value $L(A_i) = i$ for $i = 1,2,\ldots,2D_I^f(0)$. In other words, for every NFS(R_f,f) $= (u_1,\ldots,u_{R_f-1}, u_{R_f} = f)$, $u_j(A_i)$ (for $j=1,\ldots,R_f$) for each of these A_i's is uniquely determined by the equation $\sum_{j=1}^{R_f} 2^{R_f-j} u_j(A_i) = i$.

Now let us consider a sequence of functions $(u_1',\ldots,u_{R_f-1}',u_{R_f})$ which is a permutation of $(u_1,\ldots,u_{R_f-1},u_{R_f})$. Let k be the smallest number satisfying $u_k' \neq u_k$. Then for a particular vertex A_i such that $i = \sum_{j=1}^{k} 2^{R_f-j}$, $u_1(A_i)=\ldots=u_k(A_i)=1$ and $u_{k+1}(A_i)=\ldots=u_{R(f)}(A_i)=0$ must hold. This means that $u_k'(A_i)$ must have the value 0. But this contradicts the property proved in the previous paragraph that, for each NFS(R_f,f), $u_j(A_i)$ for each j including j=k has the unique value determined by $\sum_{j=1}^{R_f} 2^{R_f-j} u_j(A_i)=i$. Therefore, no permutation of

$(u_1,\ldots,u_{R_f})$ can be an NFS(R_f,f). This leads to the theorem statement that if f has more than one different NFS(R_f,f), every pair of them are distinguishable.

Q.E.D.

The above two lemmas lead to the following theorem which characterizes the necessary and sufficient condition for a function f to have a unique NFS(R_f,f).

Theorem 4.1 - A switching function f has a unique NFS(R_f,f) if and only if $D^f_0(A) + D^f_I(A) = 2^{R_f-1} - 1$ for every vertex A of C_n.

Proof - The "if" part of the theorem is easy to prove. Let $C_n(u_1,\ldots,u_{R_f} = f)$ be the labeled n-cube corresponding to an arbitrary NFS(R_f,f). According to Theorem 3.2 and Corollary 3.1, $L^f_{mn}(A) = 2D^f_I(A) + f(A)$ is the minimum possible value for $\ell(A; u_1,\ldots, u_{R_f})$ for every vertex $A \in C_n$. Similarly according to Theorem 3.3 and Corollary 3.3, $L^f_{mx}(A) = 2^{R_f} - 2(D^f_0(A) + 1) + f(A)$ is the maximum possible value for $\ell(A; u_1,\ldots,u_{R_f})$ for every vertex $A \in C_n$. Since $D^f_0(A) + D^f_I(A) = 2^{R_f} - 1$ holds for every vertex $A \in C_n$, $L^f_{mx}(A) = L^f_{mn}(A)$ holds, and therefore $\ell(A; u_1,\ldots,u_{R_f})$ is unique for every vertex $A \in C_n$. This proves the "if" part of the theorem.

The "only if" part of the theorem can be proved, using Lemmas 4.1 and 4.2. First, let us assume $D^f_0(0) + D^f_I(0) \equiv D^f_0(I) + D^f_I(I) \equiv D^f_0(I) \equiv D^f_I(0) \neq 2^{R_f-1} - 1$ holds. By Lemma 4.1, f has at least two distinguishable NFS(R_f,f)'s, so f does not have a unique NFS(R_f,f). Therefore, if f has a unique NFS(R_f,f), $D^f_0(I) \equiv D^f_I(0) = 2^{R_f-1} - 1$ must hold.

Next, let us assume that $D_0^f(A) + D_I^f(A) = 2^{R_f-1} - 1$ does not hold for some vertex $A \varepsilon C_n$, but $D_0^f(I) = D_I^f(0) = 2^{R_f-1} - 1$ holds. Then from Theorems 3.2 and 3.3 $L_{mn}^f(A) \neq L_{mx}^f(A)$ must hold, which means that there are at least two different NFS(R_f,f)'s generated by MNL and MXL. By Lemma 4.2, these two NFS(R_f,f)'s must be distinguishable. Therefore, f does not have a unique NFS(R_f,f).

Summarizing the above, the theorem must hold.

Q.E.D.

From this theorem and the definitions of $D_0^f(A)$ and $D_I^f(A)$, the following theorems can be easily proved.

<u>Theorem 4.2</u> - A function f has a unique NFS(R_f,f) (hence a unique negative gate network), if and only if for each vertex A in $C_n(f)$ there exists a directed path from vertex I to vertex 0 through A such that the number of inversions along this path is $2^{R_f-1} - 1$.

<u>Proof</u> - The "if" part can be proved as follows. By definition, $D_I^f(A) + D_0^f(A)$ must be greater than or equal to the number of inversions on any directed path from vertex I to vertex 0 through vertex A. Therefore, by assumption of the theorem, $D_I^f(A) + D_0^f(A) \geq 2^{R_f-1} - 1$ must hold. However, $D_I^f(A) + D_0^f(A) \leq D_I^f(0) \leq 2^{R_f-1} - 1$ must hold from the definition of $D_I^f(0)$ and (3.5), so $D_I^f(A) + D_0^f(A) = 2^{R_f-1} - 1$ must hold for every $A \varepsilon C_n$. By Theorem 4.1, f must have a unique solution.

Next let us prove the "only if" part of the theorem. Since f has a unique solution, by Theorem 4.1, $D_I^f(A) + D_0^f(A) = 2^{R_f-1} - 1$ must hold for every $A \varepsilon C_n$. From the definitions of $D_I^f(A)$, $D_0^f(A)$ and $D_I^f(0)$, it is obvious that there exists a directed path from I to 0 through A such that the number of inversions along this path is $2^{R_f-1} - 1$.

Q.E.D.

Theorem 4.3 - A function f has a unique NFS(R_f,f) (hence a unique minimum negative gate network) if and only if $L^f_{mn}(A) = L^f_{mx}(A)$ holds for every $A \varepsilon C_n$.

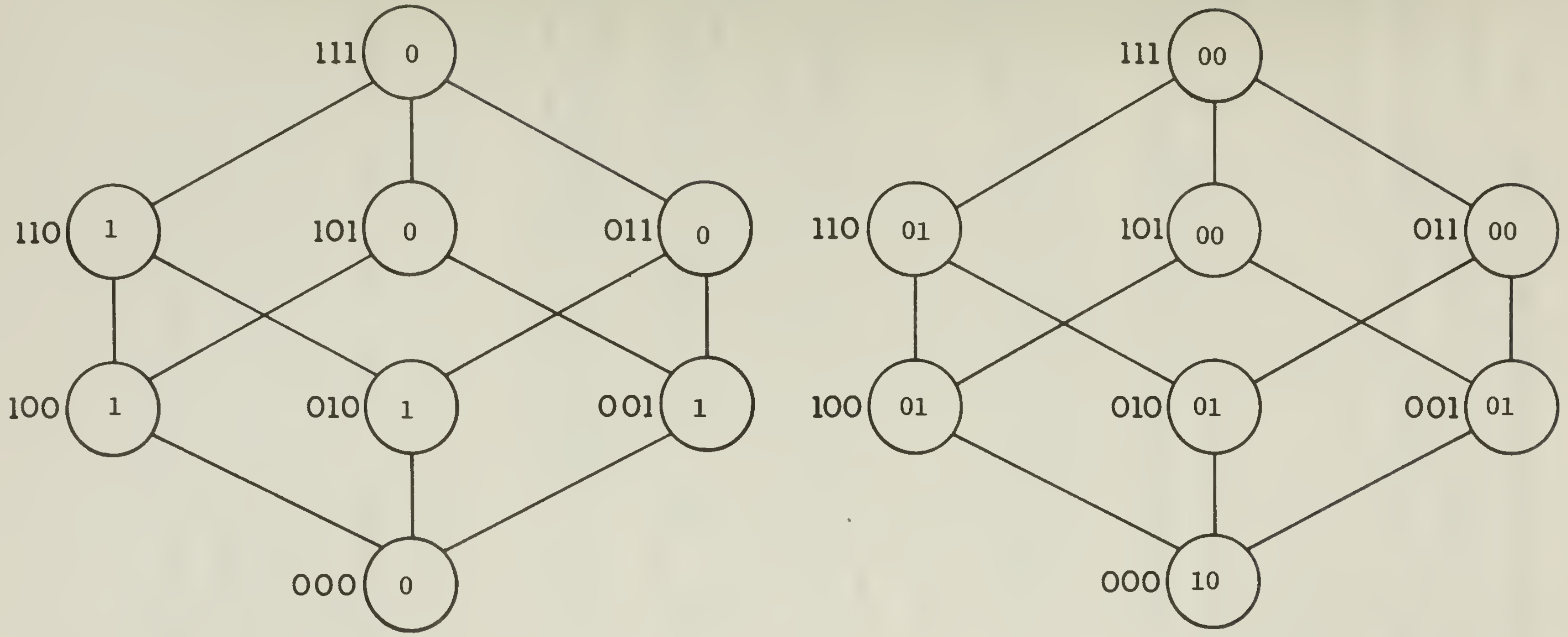
Proof - From Theorem 3.2, $L^f_{mn}(A) = 2D^f_I(A) + f(A)$ holds for every $A \varepsilon C_n$. From Theorem 3.3, $L^f_{mx}(A) = 2^{R_f} - 2(D^f_0(A) + 1) + f(A)$ holds for every $A \varepsilon C_n$. Therefore, $L^f_{mn}(A) = L^f_{mx}(A)$ implies $2D^f_I(A) = 2^{R_f} - 2(D^f_0(A) + 1)$ for every $A \varepsilon C_n$. This is equivalent to $D^f_I(A) + D^f_0(A) = 2^{R_f-1} - 1$ for every $A \varepsilon C_n$. Consequently, by Theorem 4.1, this theorem has been proved.

Q.E.D.

The condition for a function to have a unique minimum negative gate network is very tight. In other words, the functions which have unique minimum negative gate networks are only a small portion of all the functions of n variables when $n \geq 3$ (when n=1, or 2, it may not be true).

Example 4.2 - The three-variable function $f = x_1\bar{x}_3 \vee x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3$ shown in Fig. 4.1(a) has a unique NFS(R_f,f) (hence a unique minimum negative gate network). Fig. 4.1(b) shows the unique 2-bit labeling for function f. The corresponding NFS(R_f,f) is

$(u_1 = \overline{x_1 \vee x_2 \vee x_3}\ ,\ u_2 = \overline{u_1 \vee (x_1 \vee x_2)x_3} = f)$.



(a) $C_3(f)$ for $f = x_1\bar{x}_3 \vee x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3$.

(b) The unique 2-bit labeling for f.

Fig. 4.1 Example 4.2.

Let us consider parity functions $x_1 \oplus x_2 \oplus \ldots \oplus x_n$ and $x_1 \oplus x_2 \oplus \ldots \oplus x_n \oplus 1$ in regard to the uniqueness of their minimum negative gate networks. The following two theorems give necessary and sufficient conditions for a parity function to have a unique minimum negative gate network.

Theorem 4.4 - The odd parity function of n variables, $f_o = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ has a unique minimum negative gate network if and only if $n = 2^{k+1} - 2$ or $2^{k+1} - 3$ for $k = 1,2,\ldots$.

Proof - In $C_n(f_o)$, every vertex A of weight w has a label $f_o(A) = 0$ if w is even or a label $f_o(A) = 1$ if w is odd. Therefore, in every directed path from vertex I to vertex 0 there are $n/2$ or $(n+1)/2$ inverse edges, depending on whether n is even or odd, respectively. This means that $D_I^f(A) + D_O^f(A) = n/2$ or $(n+1)/2$ for every $A \in C_n$. Since $R_{f_o} = \lceil \log_2(D_I^{f_e}(O) + 1) \rceil + 1 = \lceil \log_2 (\frac{n}{2} + 1) \rceil + 1$ holds if n is even, or $R_{f_o} = \lceil \log_2 (\frac{n+1}{2} + 1) \rceil + 1$ holds if n is odd, it follows that $n = 2^{k+1} - 2$ or $n = 2^{k+1} - 3$ is the necessary and sufficient condition for f_o to have a unique minimum negative gate network.

Q.E.D.

Theorem 4.5 - The even parity function of n variables, $f_e = x_1 \oplus x_2 \oplus \ldots \oplus x_n \oplus 1$ has a unique minimum negative gate network if and only if $n = 2^{k+1} - 2$ or $2^{k+1} - 1$ for $k = 1,2,\ldots$.

Proof - The proof is similar to that of Theorem 4.2. In $C_n(f_e)$, every vertex A of weight w has a label $f_e(A)=1$ if w is even or

$f_e(A)=0$ if w is odd. Therefore in every directed path from vertex I to vertex 0 there are $n/2$ or $(n-1)/2$ inverse edges, depending on whether n is even or odd, respectively. This means that

$D_0^{f_e}(A) + D_I^{f_e}(A) = n/2$ or $(n-1)/2$ for every $A \varepsilon C_n$. Since

$R_{f_e} = \lceil \log_2(D_I^{f_e}(0) + 1) \rceil + 1 = \lceil \log_2(\frac{n}{2} + 1) \rceil + 1$ holds if n is even,

or $R_{f_e} = \lceil \log_2(\frac{n-1}{2} + 1) \rceil + 1$ holds if n is odd, it follows that

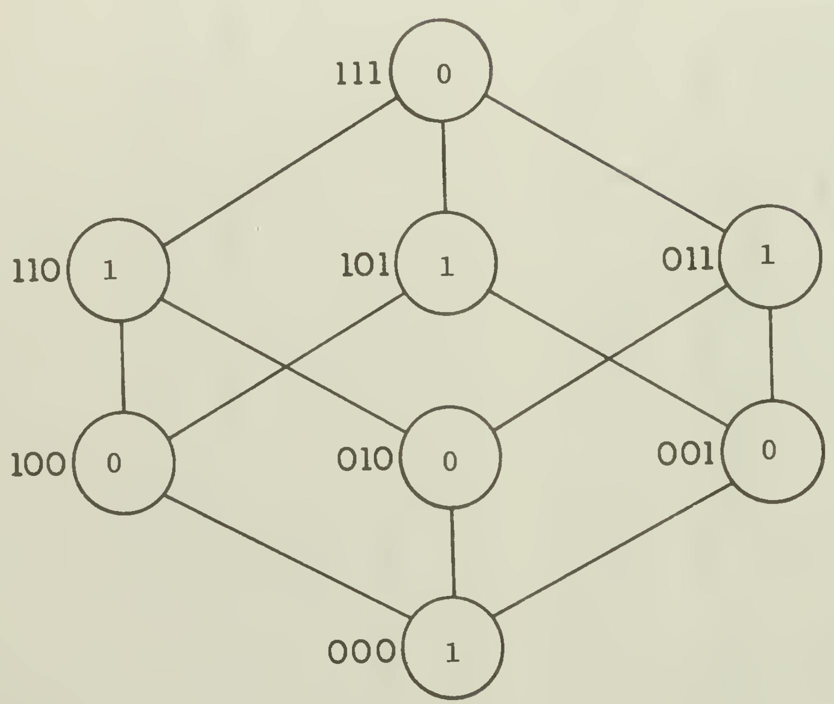
$n = 2^{k+1} - 2$ or $n = 2^{k+1} - 1$, for $k = 1,2,\ldots,$ is the necessary and sufficient condition for f_e to have a unique minimum negative gate network.

Q.E.D.

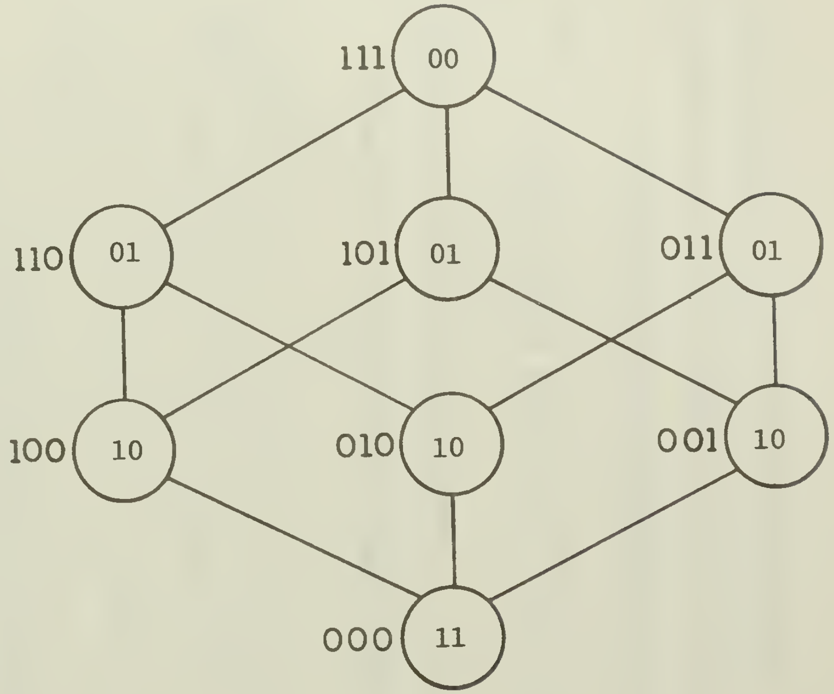
<u>Example 4.3</u> - 3-variable even-parity function

$f_e = x_1 \oplus x_2 \oplus x_3 \oplus 1$ shown in Fig. 4.2(a) has a unique NFS(R_f, f) corresponding to the unique 2-bit, inverse-edge-free labeling for f_e as shown in Fig. 4.2(b). The corresponding NFS(R_{f_e}, f_e) is

$(u_1 = \overline{x_1x_2 \vee x_1x_3 \vee x_2x_3}\ ,\ u_2 = \overline{u_1x_1 \vee u_1x_2 \vee u_1x_3 \vee x_1x_2x_3} = f_e)$.



(a) $C_3(f_e)$ for $f_e = x_1 \oplus x_2 \oplus x_3 \oplus 1$.



(b) The unique 2-bit, inverse-edge-free labeling for f_e.

Fig. 4.2 Example 4.3.

5. SYNTHESIS OF MOS CELLS

In the previous sections, the synthesis procedures of logical networks with a minimum number of negative gates were discussed. Given a switching function f of n variables, the synthesis procedures can generate minimum negative gate networks realizing the given function by specifying the function u_i realized by each negative gate g_i in the network. These functions u_i ($1 \le i \le R_f$) are specified in the form of a labeled n-cube $C_n(u_1,\ldots,u_R = f)$ which is equivalent to a truth table where the function u_i realized by each negative gate g_i is specified for 2^n input vectors. Although Theorem 3.1 asserts that each function u_i can be realized by a negative gate with $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ as inputs, the explicit expression of u_i as a negative function with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ is not yet given. When a logical network of MOSFETs is constructed to realize a given function, each MOS cell (i.e., negative gate), which theoretically can be made to realize an arbitrary negative function, must be properly designed. This requires each u_i to be explicitly expressed as a negative function of $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ for $1 \le i \le R_f$. This section is devoted to this discussion.

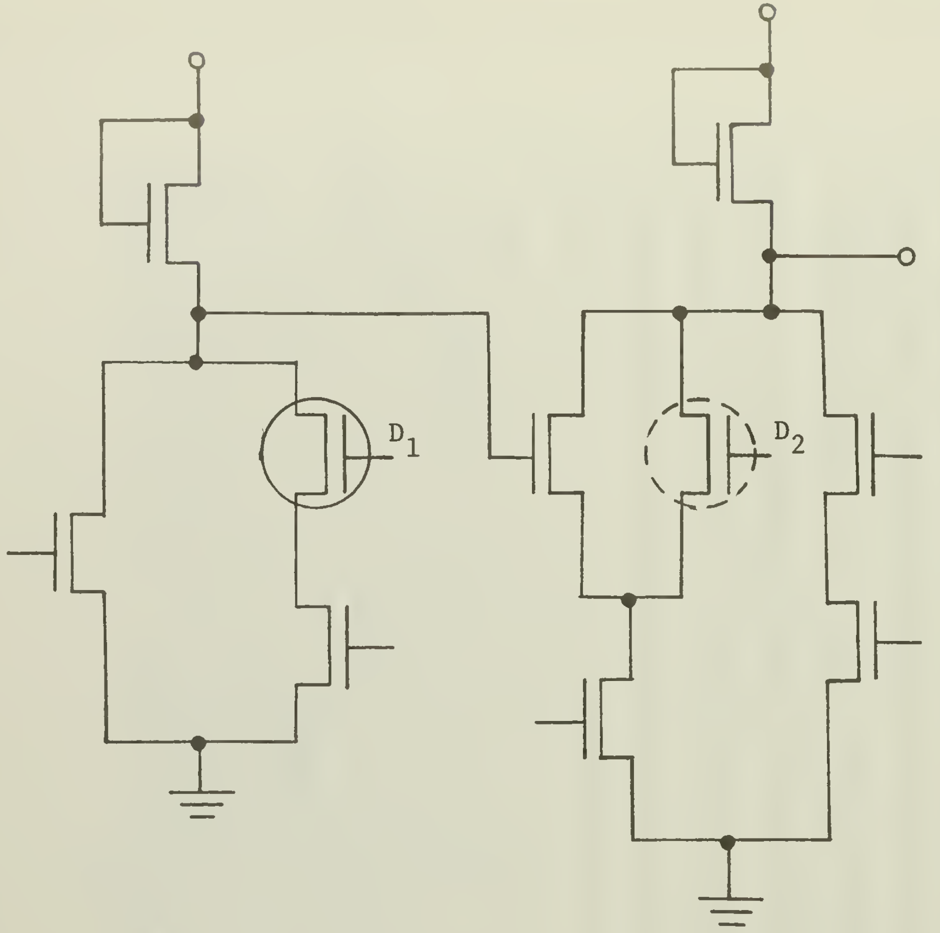
As explained in Section 2, the driver part of an MOS cell operates exactly like a relay contact network where each FET corresponds to a "make-contact" (i.e., "normally open") relay contact. Therefore, the problem of designing MOS cells for a given negative function can be treated as the problem of designing relay contact networks with only "make-contact" relays for the complement of the given function. The problem of designing relay contact networks with a minimum number of contacts has been studied by many authors. If a series-parallel type network is to be designed, a procedure proposed by E. L. Lawler [Law 64] can be applied. If a network of a general type (i.e., not necessarily series-parallel type) is to be designed, a procedure proposed by Y. Moriwaki [Mor 72] can produce an absolutely minimum network. Although these procedures are very useful in some cases, they are complicated and time-consuming even with the assistance of a high-speed computer when the size of the network is relatively large. Although these procedures, which are developed based on the assumption that both make-contacts and break-contacts are available, can be simplified to fit our special situation where only make-contacts are available, they would still be very complicated and time-consuming if minimality is to be guaranteed.

The synthesis procedure to be presented in the next section designs for a given function an "irredundant MOS network" with a minimum number of negative gates (MOS cells). In other words, the number of gates in the designed network is minimum, and if any MOSFET is removed from the network, the network will not realize the given function. (The rigorous definition of "irredundant MOS network" will be given later.) We are not concerned with whether or not each MOS cell is of series-parallel type although an MOS cell of such type is easier to design. In order to guarantee the designed network is irredundant, each MOS cell should be irredundant as defined by the following definitions.

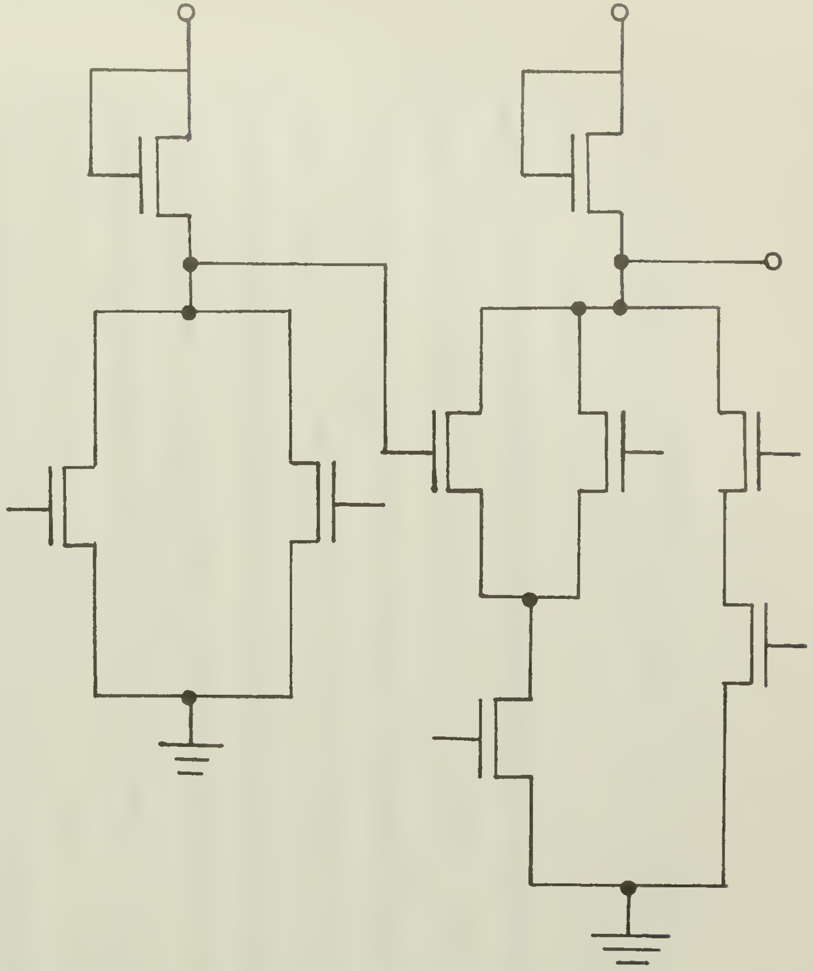
Definition 5.1 - The s-extraction of an FET from an MOS network is the replacement of this FET by a short-circuit between its source and drain. The o-extraction of an FET from an MOS network is the replacement of this FET by an open circuit between its source and drain. A single extraction of an FET is either a single s- or o-extraction of the FET, and a multiple extraction of more than one FETs is more than one simultaneous s- and/or o-extraction of these FETs.

Fig. 5.1(a) shows an MOS network. Fig. 5.1(b) shows the network obtained by the s-extraction of the FET D_1 (within the solid line circle) from the network in (a). Fig. 5.1(c) shows the network obtained by the o-extraction of the FET D_2 (within the dashed line circle) from the network in (a). Fig. 5.1(d) shows the network obtained by the simultaneous s-extraction of the FET D_1 and o-extraction of the FET D_2 from the network in (a).

Definition 5.2 - An FET in an MOS network is redundant if a single extraction of this FET does not change the output values of the network corresponding to the specified input vectors. A group of FETs in an MOS network are redundant if there exists a multiple extraction of these FETs such that this multiple extraction of these FETs does not change the output values of the network corresponding to specified input vectors.

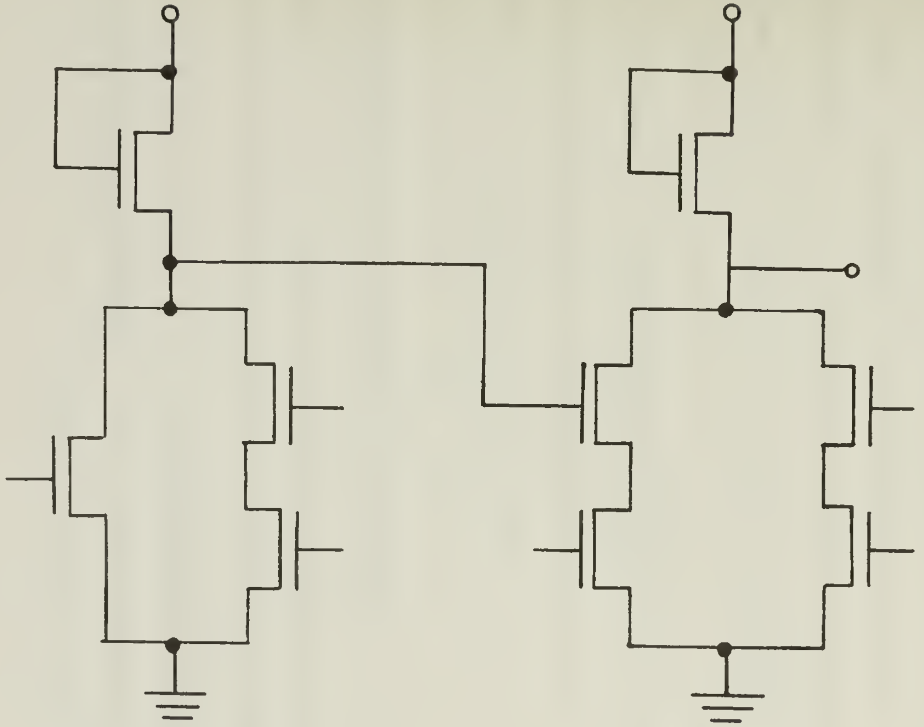


(a) Original MOS network.

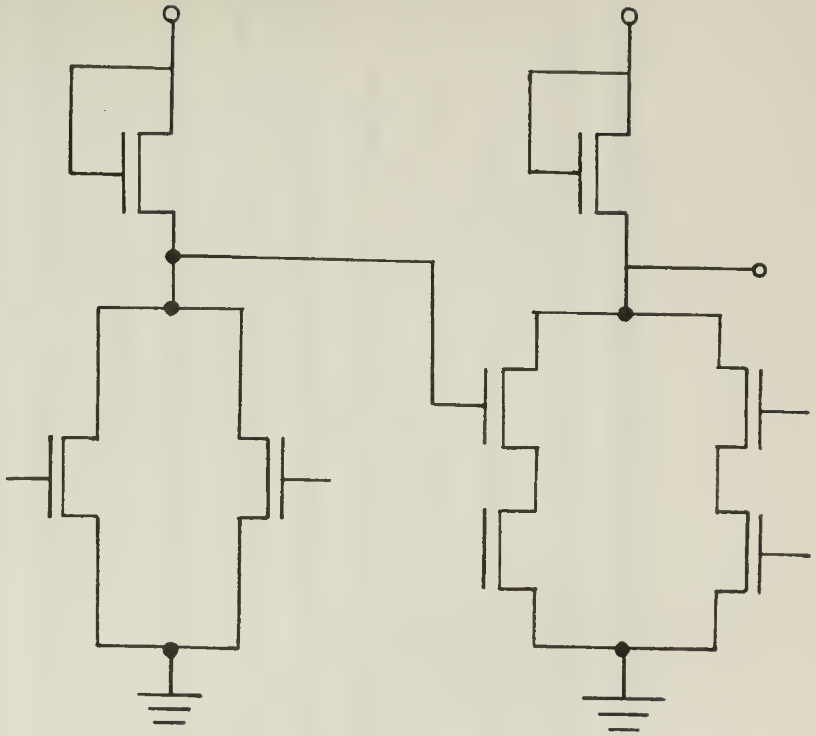


(b) Network after s-extraction of D_1 from (a)

Fig. 5.1 Examples of extractions of FETs.



(c) Network after o-extraction of D_2 from (a).



(d) Network after s-extraction of D_1 and o-extraction of D_2 from (a).

Fig. 5.1 (Continued)

<u>Definition 5.3</u> - An <u>irredundant MOS cell configuration</u> with respect to a negative (possibly incompletely specified) function f is an MOS cell realizing f (or a completion of f) that has no redundant FETs.

It should be noted that there are usually more than one irredundant MOS cell configuration for a given function.

<u>Definition 5.4</u> - An <u>irredundant MOS network</u> for a function f (possibly incompletely specified) is an MOS network realizing f (or any completion of f) that has no redundant FETs.

According to the above definitions, it is obvious that each MOS cell in an irredundant MOS network must have an irredundant MOS cell configuration with respect to the function realized by the cell.

It should be noted that even if each MOS cell in an MOS network is irredundant, the network may not be irredundant, as examples will be shown in Section 6.

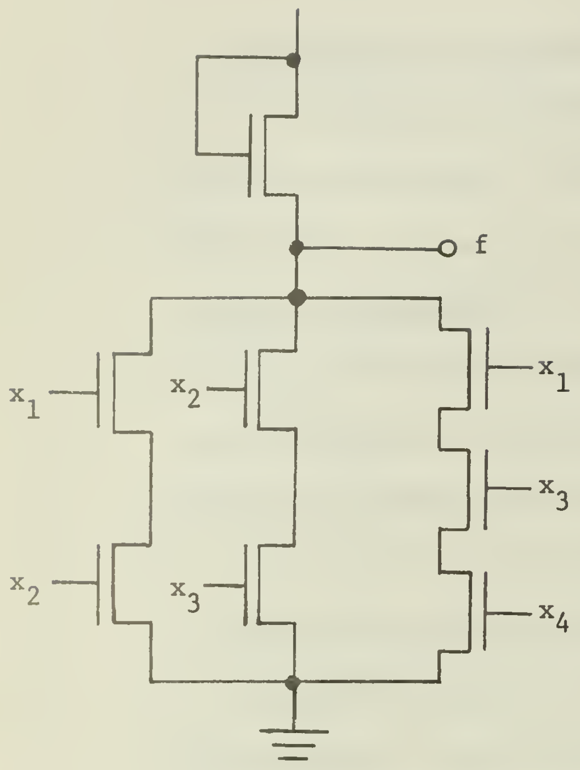
The following definition defines complemented irredundant disjunctive forms and conjunctive forms, based on which irredundant MOS cell configurations can be obtained according to Algorithms 5.1 and 5.2, respectively.

<u>Definition 5.5</u> - A <u>complemented irredundant disjunctive form</u> for a negative function f (possibly incompletely specified) is the complement of an irredundant disjunctive form for the complement of f, $\bar{f}$. A <u>complemented irredundant conjunctive form</u> for a negative function f (possibly incompletely specified) is the complement of an irredundant conjunctive form for the complement of f, $\bar{f}$.

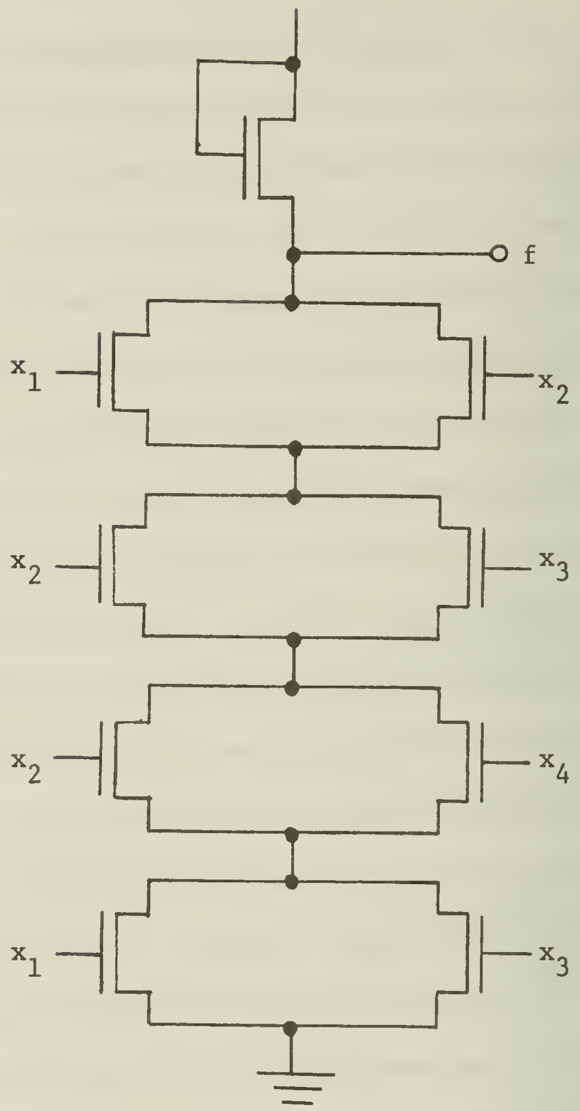
Definition 5.6 - A complemented minimum sum-of-products form for a negative function f (possibly incompletely specified) is a complemented irredundant disjunctive form for f that has the minimum number of terms and the minimum number of literals among all complemented irredundant disjunctive forms for f. A complemented minimum product-of-sums form for a negative function f (possibly incompletely specified) is a complemented irredundant conjunctive form for f that has the minimum number of alterms and the minimum number of literals among all complemented irredundant conjunctive forms of f.

Example 5.1 - Suppose a completely specified negative function $f = \overline{x_1x_2 \vee x_2x_3 \vee x_1x_3x_4}$ is given. The MOS cells shown in Fig. 5.2 (a)-(d) are irredundant MOS cell configurations for f. The MOS cell shown in Fig. 5.2(a) is obtained from the complemented minimum sum-of-products form of f, i.e., $f = \overline{x_1x_2 \vee x_2x_3 \vee x_1x_3x_4}$. Fig. 5.2(b) shows a cell obtained from the complemented minimum product-of-sums form of f, i.e., $f = \overline{(x_1 \vee x_2)(x_2 \vee x_3)(x_2 \vee x_4)(x_1 \vee x_3)}$. These two cells are not the simplest irredundant ones for f. Fig. 5.2(c) and (d) show two other simpler irredundant MOS cells for f which are obtained by factoring out common terms from (a) and (b), respectively.

Example 5.2 - An incompletely specified f of four variables is given in the truth table form ($C_4(f)$ with don't care components denoted with *) as shown in Fig. 5.3(a). Two complemented minimum sum-of-products forms of f, $f_1 = \overline{x_1x_2 \vee x_1x_3 \vee x_2x_3 \vee x_2x_4}$ and $f_2 = \overline{x_1x_2 \vee x_1x_4 \vee x_2x_3 \vee x_1x_4}$, are obtained by assigning f(1010)=0 and f(1001)=1 and also by assigning f(1010)=1 and f(1001)=0,

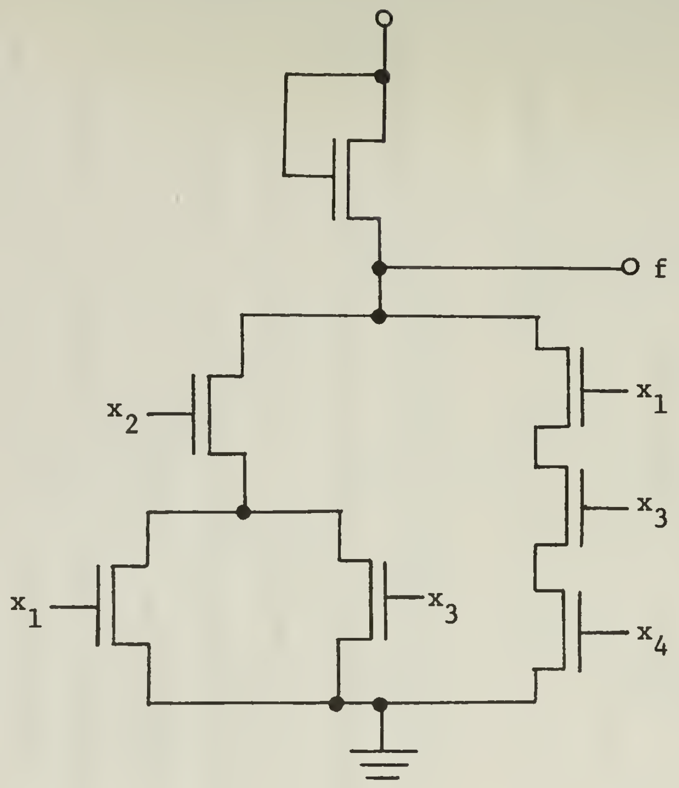


(a) An irredundant MOS cell configuration for f based on complemented minimum sum-of-products form of f.

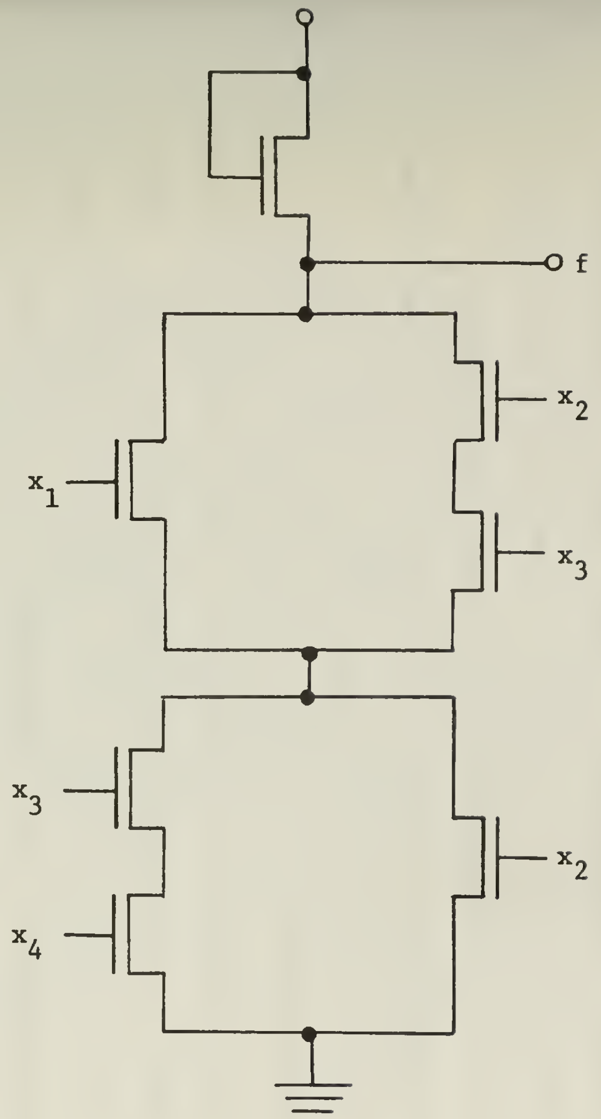


(b) An irredundant MOS cell configuration for f based on the complemented minimum product-of-sums form of f.

Fig. 5.2 Example 5.1: irredundant MOS cell configurations for $f = \overline{x_1x_2 \vee x_2x_3 \vee x_1x_3x_4}$.



(c) An irredundant MOS cell configuration for f derived from (a).

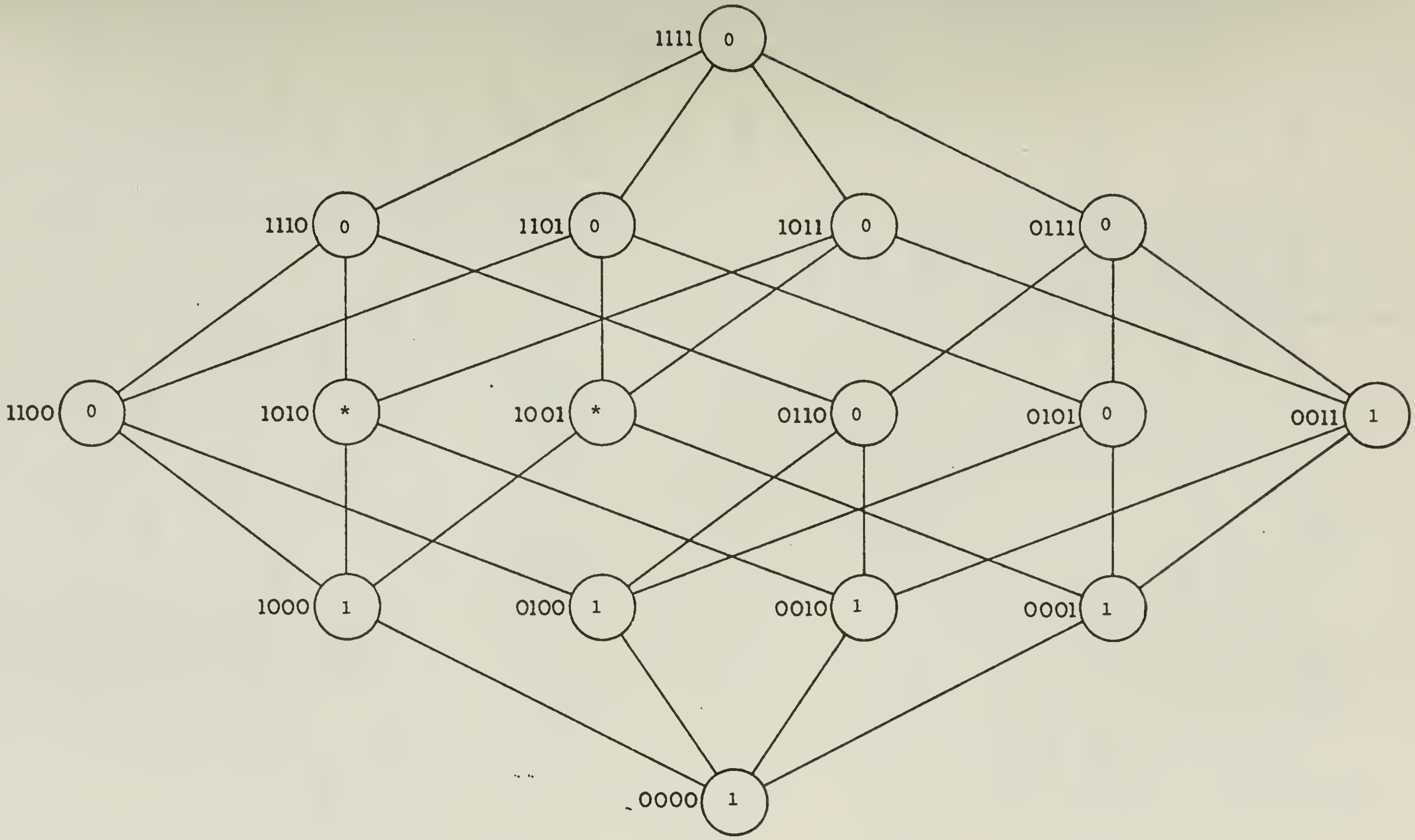


(d) An irredundant MOS cell configuration for f derived from (b).

Fig. 5.2 (Continued)

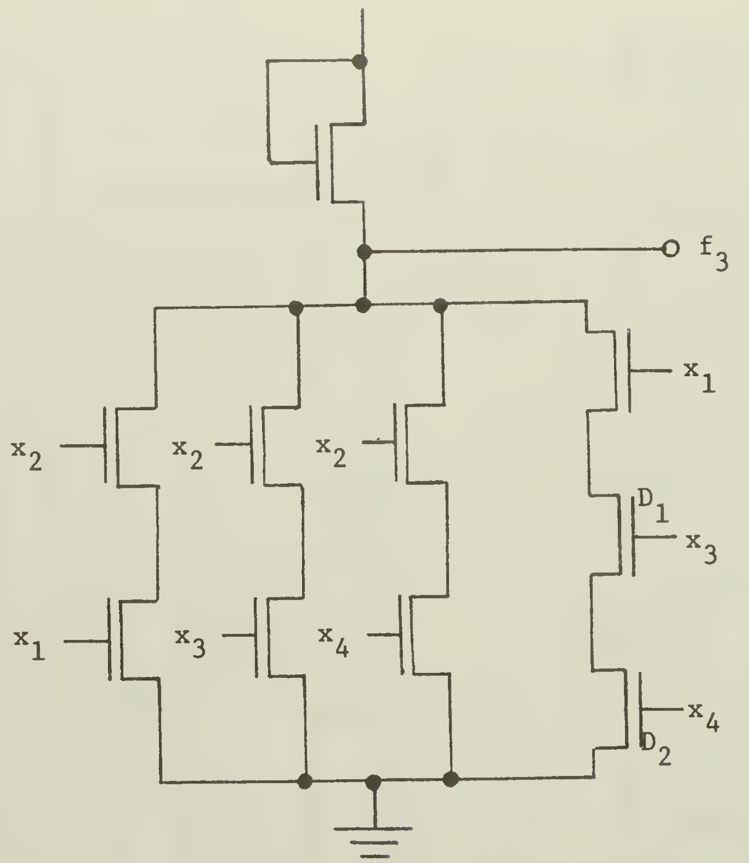
respectively. MOS cells based on these expressions or their derivatives obtained by factorization of common terms are irredundant MOS cell configurations for f (not shown). However, the direct implementation of the complemented minimum sum-of-products form for another completion of f (notice that it is the complemented minimum sum-of-products form for a completion of f but not for the incompletely specified function f), $f_3 = \overline{x_1x_2 \vee x_2x_3 \vee x_2x_4 \vee x_1x_3x_4}$ obtained by assigning f(1010)=f(1001)=1, yields a redundant MOS cell with respect to the incompletely specified function f. Fig. 5.3(b) shows this cell where the s-extraction of either FET D_1 or D_2 does not change the output values corresponding to the specified input vectors of f. The MOS cell obtained by the s-extraction of D_1 or D_2 realizes function f_2 or f_1, respectively, each of which is a negative completion of f. However, if an MOS cell is designed as shown in Fig. 5.3(c) which realize the same function as Fig. 5.3(b), it is an irredundant MOS cell configuration of f since any single or multiple extraction of FETs in (c) will make the MOS cell no longer realize a completion of f.

In general the exhaustion of all irredundant MOS cell configurations for a given negative function requires an enormous effort. As a matter of fact, if all irredundant MOS cell configurations are obtained, the ones with a minimum number of FETs among them will be the minimum MOS cell configurations for the given function. For our irredundant MOS network design procedure to be presented in the next section, an arbitrary irredundant MOS cell configuration is sufficient.

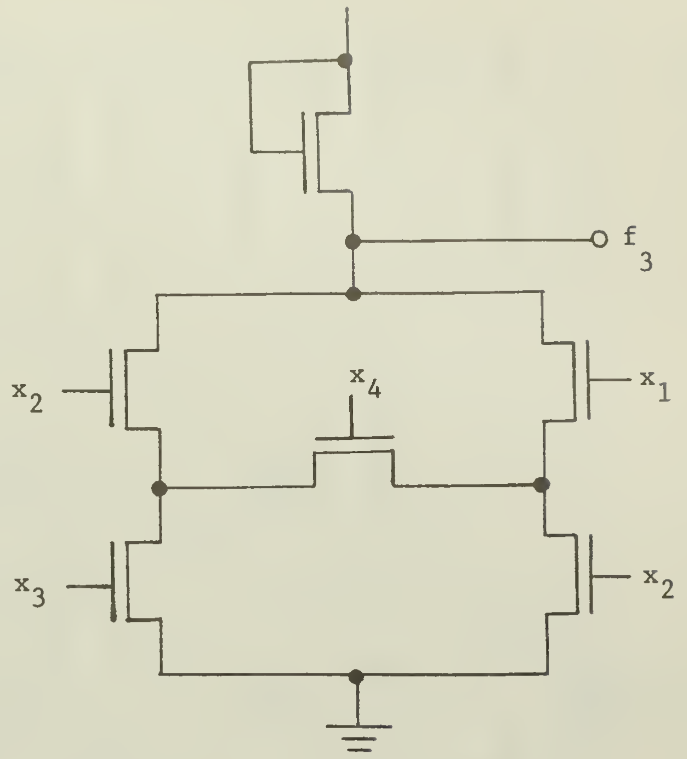


(a) Incompletely specified function f.

Fig. 5.3 Example 5.2.



(b) A redundant MOS cell configuration for f. The single s-extraction of either D_1 or D_2 from the network results in a cell realizing f_2 or f_1 which are completions of f.



(c) An irredundant MOS cell configuration for f. Although it realizes the same function f_3 as (b) does, it is an irredundant MOS cell configuration for the incompletely specified function f.

Fig. 5.3 (Continued)

The following algorithm is based on the algorithm given by T. Ibaraki [Iba 71] which obtains a complemented irredundant disjunctive form for a negative function (Definition 5.5).

Algorithm 5.1 - Derivation of a complemented irredundant disjunctive form for a given incompletely specified negative function f of n variables. (This algorithm is illustrated in Example 5.3.)

Step 1 - Obtain a subset S^f_{0*} of the set V_n of all n-dimentional binary vectors such that $X \varepsilon S^f_{0*}$ if and only if $f(x)=0$ or $*$ holds and no $y > x$ satisfying $f(y)=1$ exists.

Step 2 - Obtain Min $[S^f_{0*}]$, the set of minimum vectors of S^f_{0*}, such that $X \varepsilon$ Min $[S^f_{0*}]$ if and only if there exists no $Y \varepsilon S^f_{0*}$ satisfying $Y < X$.

Step 3 - Obtain an irredundant subset of Min$[S^f_{0*}]$, Irr$[S^f_{0*}] \subseteq$ Min$[S^f_{0*}]$ such that, for every X satisfying $f(X)=0$, there exists a $Y \varepsilon$ Irr$[S^f_{0*}]$ satisfying $Y \leq X$. (The term "irredundant" means that the removal of any vector from Irr$[S^f_{0*}]$ will cause the above condition unsatisfied.) A subset Irr$[S^f_{0*}]$ of Min$[S^f_{0*}]$ satisfying this condition will be referred to as an irredundant cover of Min$[S^f_{0*}]$. (The problem of obtaining an irredundant cover is a covering problem.)

Step 4 - For each $A = (a_1,\ldots,a_n) \varepsilon$ Irr$[S^f_{0*}]$, make the corresponding product term of input variables $P_A = x_1^{a_1} x_2^{a_2} \ldots x_n^{a_n}$ such that if $a_i = 0$, $x_i^{a_i} = 1$ (therefore x_i is not included in P_A) and if $a_i = 1$, $x_i^{a_i} = x_i$. The complement of the sum of these product terms, $f_1 = \overline{\bigvee_{A \varepsilon \mathrm{Irr}[S^f_{0*}]} P_A}$, is a complemented irredundant disjunctive form for f.

From a complemented irredundant disjunctive form, it is easy to construct an irredundant MOS cell configuration for f. For each term P_A in f_1, make a serial connection of FETs such that corresponding to each literal in P_A there is an FET with the input represented by the literal. Then, parallelly connect all these serially connected FETs. The resulting MOS configuration constitutes the driver part of the irredundant MOS cell configuration designed for f.

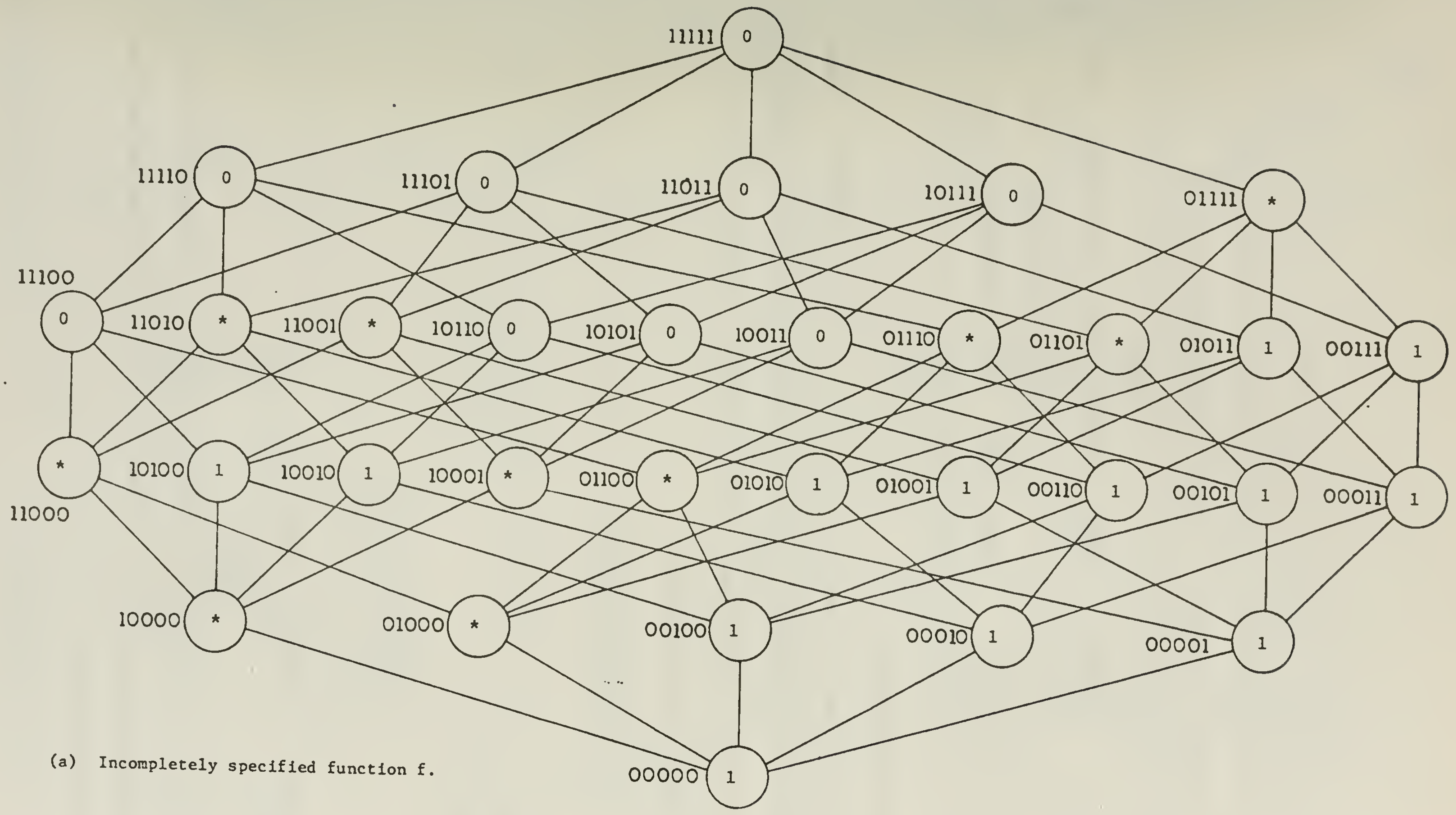
Steps 1 and 2 of Algorithm 5.1 involve only simple calculations. Step 3 requires the solution of a covering problem which is relatively simple since any irredundant cover will be sufficient for our purpose. Step 3 is necessary since f is generally an incompletely specified function (see Example 5.3 to be presented later). If f is a completely specified negative function, it will have only one complemented irredundant disjunctive form. In such a case Step 3 is unnecessary because $\mathrm{Irr}[S^f_{0*}] = \mathrm{Min}[S^f_{0*}]$. In Step 3, if a minimum cover is obtained the resulting configuration will contain a smaller number of FETs. (This may not be true, if factorization is considered.) Step 4 of this algorithm is trivial.

The validity of this algorithm can be easily proved. First to prove that the constructed MOS cell realizes a completion of f, we need to prove that for every specified false vector, there is a path between the two terminals of the driver part of the cell. This is obvious since Step 3 guarantees that for each vector A such that $f(A) = 0$, there is a vector B in $\mathrm{Irr}[S^f_{0*}]$ such that $B \leq A$. Therefore, when input A is applied to the cell, the serially connected FETs corresponding to B forms a conductive path. Similarly for each vector A such that $f(A) = 1$, no vector $B \varepsilon [\mathrm{Irr}\ S^f_{0*}]$ satisfies $B \leq A$ according to Steps 1 and 2. Therefore, when A is applied, there will be no path in the driver part of the constructed cell. This proves that the constructed MOS cell realizes a completion of f. Next, we need to prove that the MOS cell obtained is an irredundant MOS cell configuration for f. If it is not, there must exist one or more FETs such that the extractions of them do not affect the realization of f. Let us first consider the case where one FET is redundant. According to the manner in which the cell is constructed, the s-extraction of an FET which corresponds to literal x_i in a term P_A means the replacement of $A \varepsilon \mathrm{Irr}[S^f_{0*}]$ by a vector $B \not\varepsilon \mathrm{Irr}[S^f_{0*}]$ such that $b_i=0$ $(a_i=1)$ and $b_j=a_j$ for $j \neq i$. Since $B \not\varepsilon \mathrm{Min}[S^f_{0*}]$ (otherwise $A \varepsilon \mathrm{Irr}[S^f_{0*}]$ can not hold), it is clear that there must exist a vector B' such that $B' \geq B$ and $f(B') = 1$. However, when B' is applied to the cell obtained by the s-extraction of the FET, a conductive path from the output terminal to ground will appear which results in the output value 0 since $B' \geq B$ and the s-extraction of the FET causes the replacement of term P_A by term P_B in the corresponding expression. Therefore, no s-extraction of any FET can be performed without changing the output values of f corresponding to some specified inputs. On the other hand, the o-extraction of an FET corresponding to any literal in term P_A from the cell means the removal of the vector A from $Irr[S^f_{0*}]$. Since $Irr[S^f_{0*}]$ is, according to Step 3, an irredundant cover of $Min[S^f_{0*}]$ satisfying that for every X satisfying $f(X)=0$, there exists a $Y \in Irr[S^f_{0*}]$ satisfying $Y \leq X$, the removal of term A from $Irr[S^f_{0*}]$ means that for some vector B satisfying $B > A$ and $f(B)=0$, there exists no $Y \in Irr[S^f_{0*}]$ satisfying $Y < B$. This, in turn, means that the resulting MOS cell after the o-extraction will produce an output "1" for input B. Consequently, the designed MOS cell by Algorithm 5.1 is an irredundant MOS cell configuration for f.

Sometimes the number of FETs in an MOS cell designed by Algorithm 5.1 can be reduced by factoring out common literals in the expression obtained in Step 4. Obviously, this factorization does not change the irredundancy of the MOS cell.

Example 5.3 - A switching function f is specified as shown in Fig. 5.4(a).

By Step 1 of Algorithm 5.1, S^f_{0*} = {(11111), (11110), (11101), (11011), (10111), (01111), (11100), (11010), (11001), (10110), (10101), (10011), (01110), (01101), (11000), (10001), (01100)}. Then Step 2



(a) Incompletely specified function f.

Fig. 5.4 Example 5.3.

calculates $\mathrm{Min}[S^f_{0*}] = \{(11000), (10001), (01100), (10110)\}$. Since (10110) and (10001) must be included in every irredundant subset $\mathrm{Irr}[S^f_{0*}]$ of $\mathrm{Min}[S^f_{0*}]$ defined in Step 3 of Algorithm 5.1, the only irredundant $\mathrm{Irr}[S^f_{0*}]$'s are $\mathrm{Irr}[S^f_{0*}]_1 = \{(11000), (10001), (10110)\}$ and $\mathrm{Irr}[S^f_{0*}]_2 = \{(10001), (01100), (10110)\}$. The complemented irredundant disjunctive forms for f are $f_1 = \overline{x_1x_2 \vee x_1x_5 \vee x_1x_3x_4}$ and $f_2 = \overline{x_1x_5 \vee x_2x_3 \vee x_1x_3x_4}$, respectively. The corresponding irredundant MOS cell configurations for f are shown in (b) and (c) of Fig. 5.4, respectively.

Although these two expressions are the only complemented minimum sum-of-products forms for f, other irredundant MOS cells with less FETs can be derived from them by factorization. The MOS cell shown in Fig. 5.4(d) corresponds to

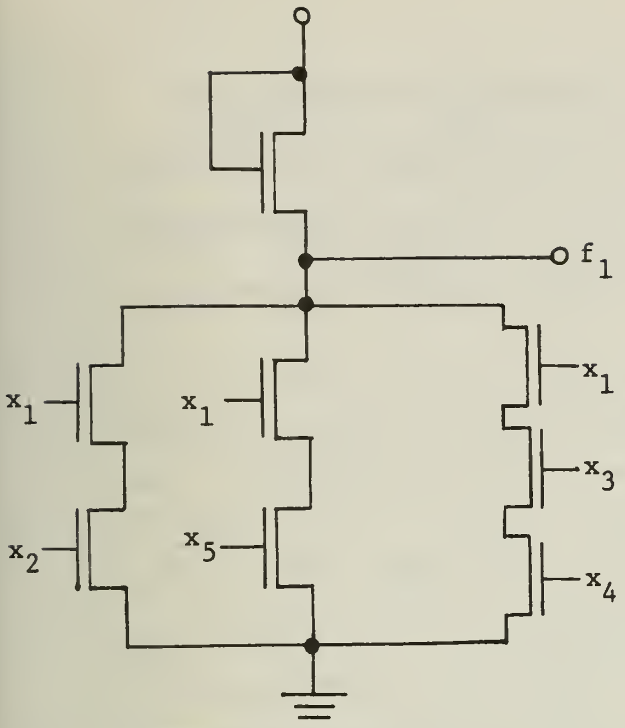
$$f_1 = \overline{x_1\ (x_2 \vee x_5 \vee x_3x_4)}$$

which uses only five driver FETs. The MOS cell shown in Fig. 5.4(e) corresponds to

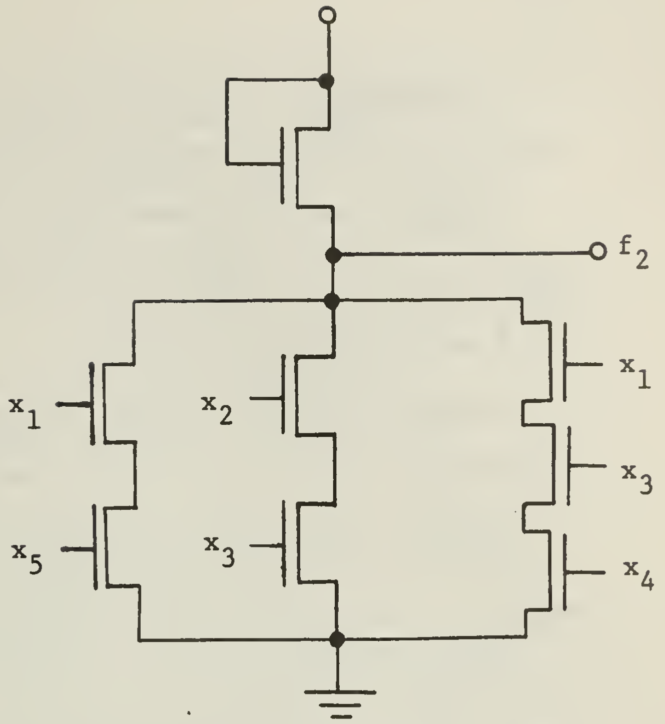
$$f_2 = \overline{x_1\ (x_5 \vee x_3x_4) \vee x_2x_3}$$

which requires one more FET than (d).

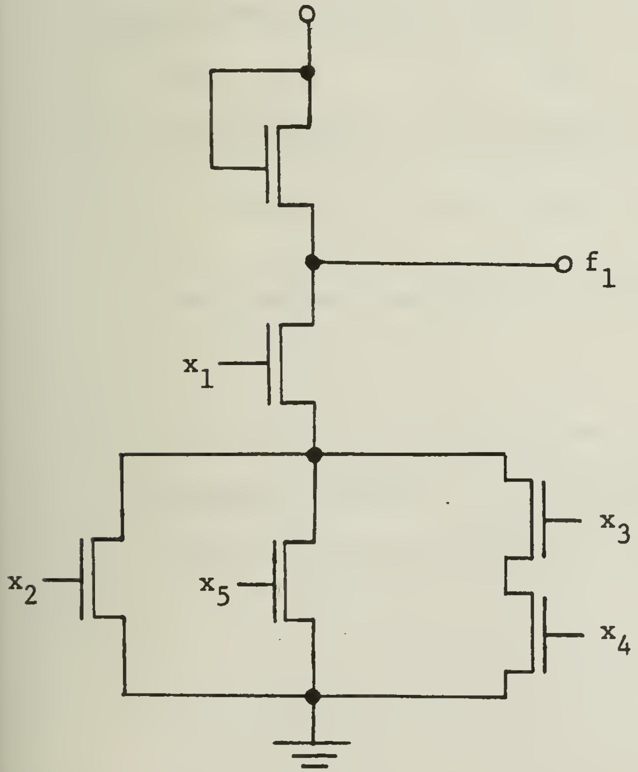
By the dual property, it is easy to obtain a complemented irredundant conjunctive form for a given function. The algorithm and an example is shown next.



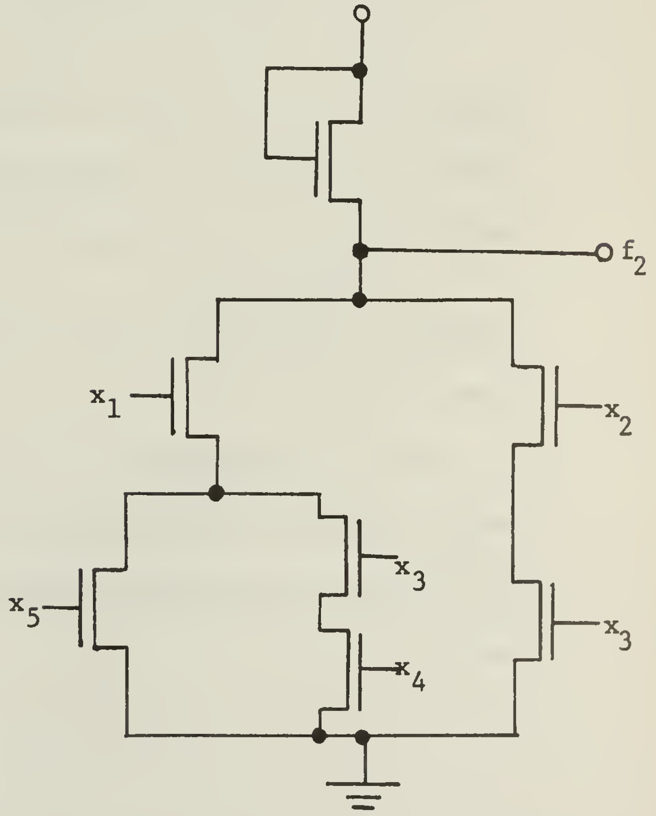
(b) Irredundant MOS cell for f based on f_1.



(c) Irredundant MOS cell for f based on f_2.



(d) Irredundant MOS cell for f based on factorization of f_1.



(e) Irredundant MOS cell for f based on factorization of f_2.

Fig. 5.4 (Continued)

Algorithm 5.2 - Derivation of a complemented irredundant conjunctive form for an incompletely specified function f of n variables.

Step 1 - Obtain a subset S^f_{1*} of the set V_n of all n-dimensional binary vectors such that $X \varepsilon S^f_{1*}$ if and only if f(X)=1 or * and no $Y < X$ satisfying f(Y)=0 exists.

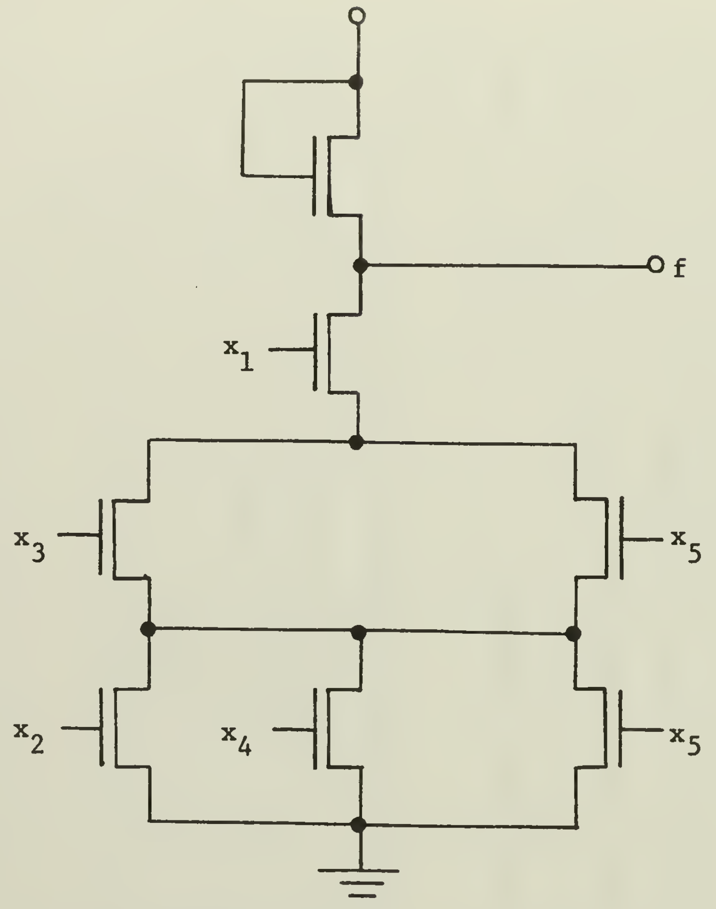
Step 2 - Obtain $\mathrm{Max}[S^f_{1*}]$, the maximum vectors of S_{1*}, such that $X \varepsilon \mathrm{Max}[S^f_{1*}]$ if and only if there exists no $Y \varepsilon S^f_{1*}$ satisfying $Y > X$.

Step 3 - Obtain an irredundant subset of $\mathrm{Max}[S_{1*}]$, $\mathrm{Irr}[S^f_{1*}] \subseteq \mathrm{Max}[S_{1*}]$ such that for every X satisfying f(x)=1, there exists a $Y \varepsilon \mathrm{Irr}[S^f_{1*}]$ satisfying $Y \geq X$ and that the deletion of any vector from $\mathrm{Irr}[S^f_{1*}]$ will cause the above condition not satisfied for some f(X)=1. ($\mathrm{Irr}[S^f_{1*}]$ is called an irredundant cover of $\mathrm{Max}[S^f_{1*}]$.)

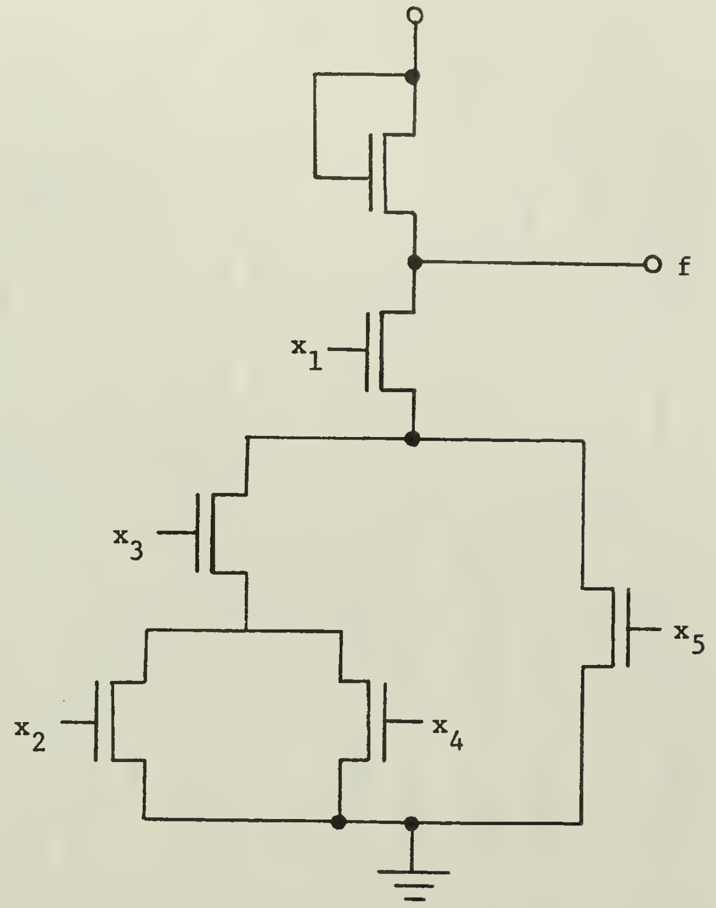
Step 4 - For each $A = (a_1,\ldots,a_n) \varepsilon \mathrm{Irr}[S^f_{1*}]$ make the corresponding alterm of input variables $Q_A = \bar{a}_1 x_1 \;\; \bar{a}_2 x_2 \;\; \ldots \;\; \bar{a}_n x_n$ such that if a_i=0 then $\bar{a}_i$=1 (therefore x_i appears in Q_A. Otherwise x_i is not in Q_A). The complement of the conjunction of all these alterms, $\overline{\bigwedge_{A \varepsilon \mathrm{Irr}[S^f_{1*}]} Q_A}$, is a complemented irredundant conjunctive form for f.

From the complemented irredundant conjunctive form of f, it is easy to construct an irredundant MOS cell for f. The dual procedure of the one discussed after Algorithm 5.1 can be used and is omitted here.

Example 5.4 - Consider the function f given in Example 5.3. Step 1 of Algorithm 5.2 obtains S^f_{1*} = {(00000), (10000), (01000), (00100), (00010), (00001), (11000), (10100), (10010), (10001), (01100), (01010), (01001),(00110), (00101), (00011), (11010), (01110), (01101), (01011), (00111), (01111)}. Then Step 2 calculates $\mathrm{Max}[S^f_{1*}]$ = {(11010), (11001), (10100), (01111)}. In Step 3, $\mathrm{Irr}[S^f_{1*}]$ = {(11010), (10100), (01111)} is obtained as the only irredundant cover. Therefore, $f_1 = \overline{x_1(x_3 \vee x_5)(x_2 \vee x_4 \vee x_5)} = \overline{x_1(x_3(x_2 \vee x_4) \vee x_5)}$. The irredundant MOS cell corresponding to the above two expressions are shown in Fig. 5.5(a) and (b), respectively. The second cell happens to be an MOS cell with a minimum number of FETs (since f depends on all five input variables and each variable requires at least one FET).



(a) Irredundant MOS cell for f based on $f_1 = \overline{x_1(x_3 \vee x_5)(x_2 \vee x_4 \vee x_5)}$.



(b) Irredundant MOS cell for f based on $f_1 = \overline{x_1(x_3(x_2 \vee x_4) \vee x_5)}$.

Fig. 5.5 Example 5.4.

6. DESIGN OF IRREDUNDANT MOS NETWORKS

This section will present an algorithm for the synthesis of irredundant MOS networks defined in Section 5 (Definition 5.4). First we will show, in Section 6.1, that the conventional approaches for MOS network design sometimes produce redundant MOS networks. Then an outline of the algorithm to be introduced in Section 6.4 will be presented in Section 6.2. Some concepts necessary for better understanding of the algorithm will be introduced in Section 6.3. Some illustrative examples will be given in Section 6.5 to show the effectiveness of the algorithm.

6.1 Conventional Design Procedures of MOS Networks

The algorithms given in [NTK 72] and [Liu 72] design MOS networks for a given function in two separate phases:

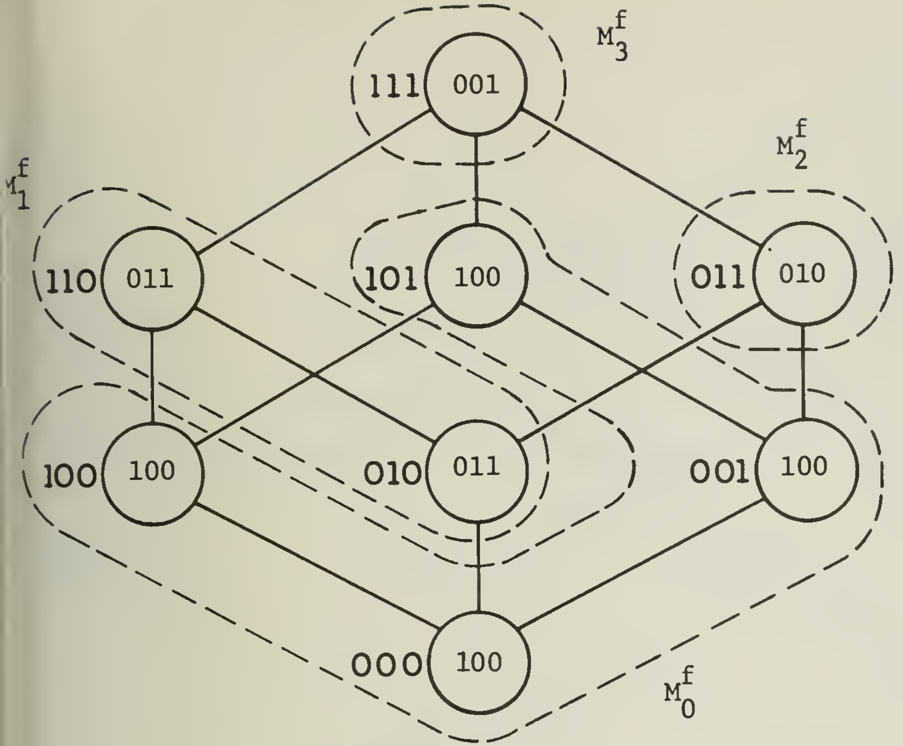
Phase I - Design an NFS(R_f,f) (a negative gate network) in which the function to be realized by each negative gate (MOS cell) is specified with respect to the external inputs.

Phase II - Design an MOS cell to realize each of the functions specified by Phase I (i.e., the design of the internal configuration of each cell).

As discussed in the previous sections, algorithms MNL and MXL design networks with a minimum number of negative gates. These algorithms and the algorithm based on the relabeling of the clustered n-cubes obtained by algorithm MNL or MXL (as demonstrated in Examples 3.3 and 3.5) correspond to Phase I in the above two-phase design procedures since the internal structure of each negative gate (MOS cell in an MOS network) is not designed by them. In order to design an MOS network, each negative gate should be realized by an MOS cell; this corresponds to Phase II of the design procedures. The design algorithms of irredundant MOS cell configurations in Section 5 can be applied for this purpose although they are different from the algorithms given by [NTK 72] and [Liu 72] which may not yield irredeundant MOS cell configurations for each negative function specified by algorithms of Phase I.

Even if each MOS cell configuration is irredundant, the conventional design approaches with two separate phases have a disadvantage that the designed MOS network may contain redundant FETs. In other words, even if we use an irredundant MOS cell design procedure for Phase II, we may not obtain an irredundant MOS network as a whole. This is demonstrated by the following example.

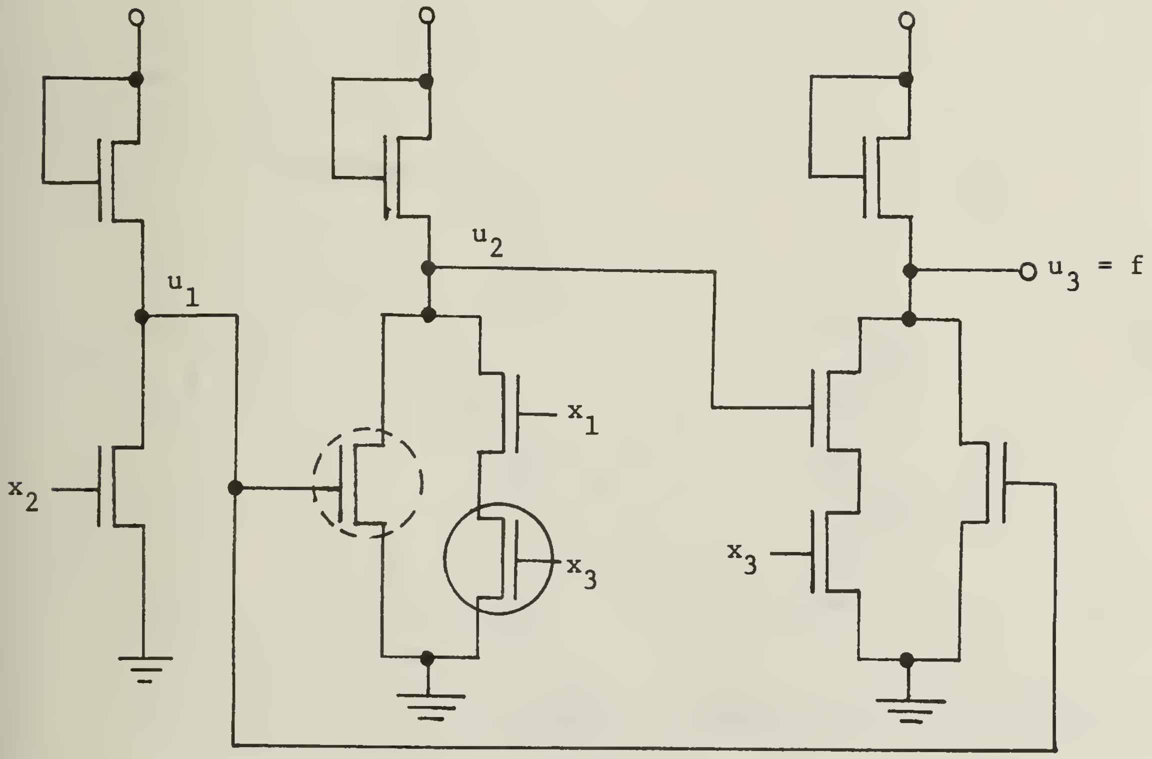
<u>Example 6.1.1</u> - Consider the five minimum NFS(3,f)'s for $f = x_1x_2 \vee x_2\bar{x}_3$ discussed in Example 3.5 (Table 3.2). If we design an irredundant MOS cell for each negative function in each NFS(3,f), we will obtain five MOS networks for f as shown in Fig. 6.1.1(a)-(e). Each figure shows the labeled 3-cube corresponding to each NFS(3,f) in Table 3.2, and an MOS network obtained by designing an irredundant MOS cell for each function in this NFS(3,f). For example, Fig. 6.1.1(a) shows the labeled 3-cube corresponding the $NFS(3,f)_1$ in Table 3.2 which is obtained based on the stratified structure of f given by [Liu 72]. The clusters of the statified structure of f are enclosed in broken



$u_1 = \bar{x}_2$

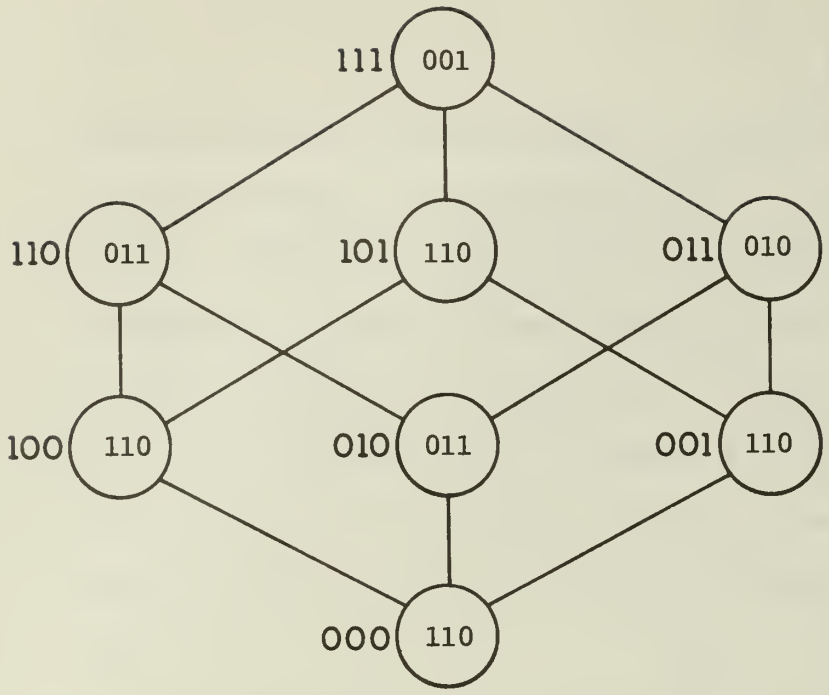
$u_2 = \overline{u_1 \vee x_1 x_3}$

$u_3 = \overline{u_1 \vee u_2 x_3} = f$



(a) Labeled 3-cube and MOS network for NFS$(3,f)_1$.

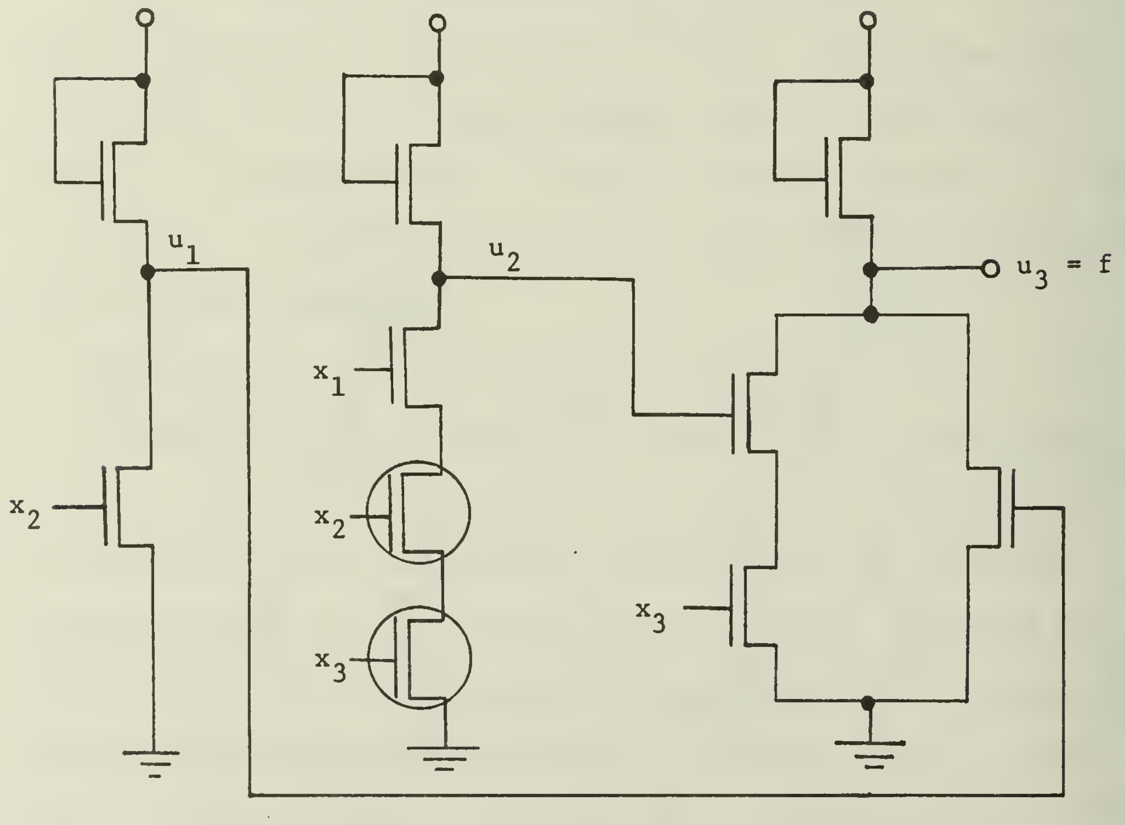
Fig. 6.1.1 Example 6.1.1: MOS networks based on stratified structure of [Liu 72] for $f = x_1 x_2 \vee x_2 \bar{x}_3$ (see Example 3.5).



$u_1 = \bar{x}_2$

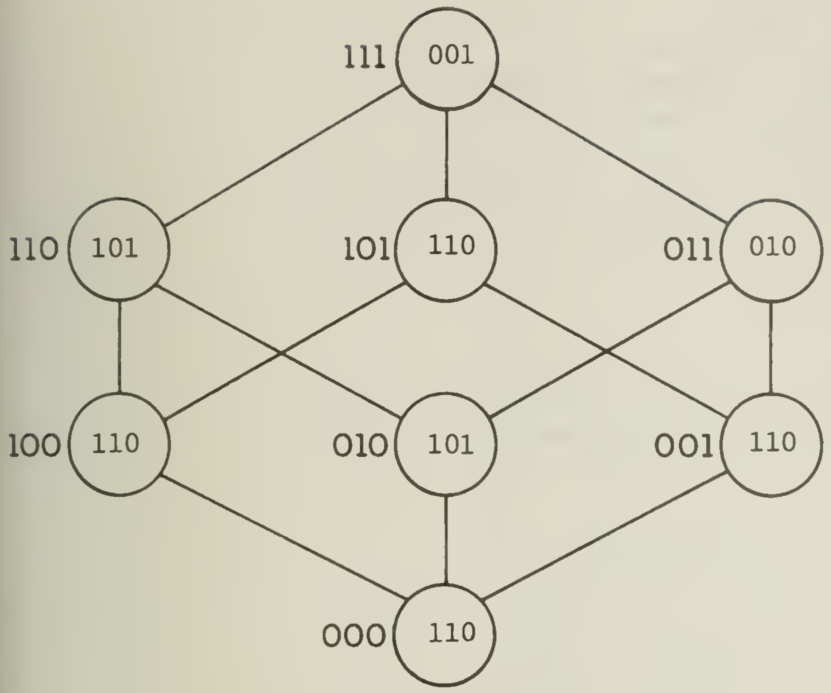
$u_2 = \overline{x_1 x_2 x_3}$

$u_3 = \overline{u_1 \vee x_3 u_2} = x_2(\bar{x}_3 \vee x_1 x_2 x_3) = f$



(b) Labeled 3-cube and MOS network for NFS$(3,f)_2$.

Fig. 6.1.1 (Continued)

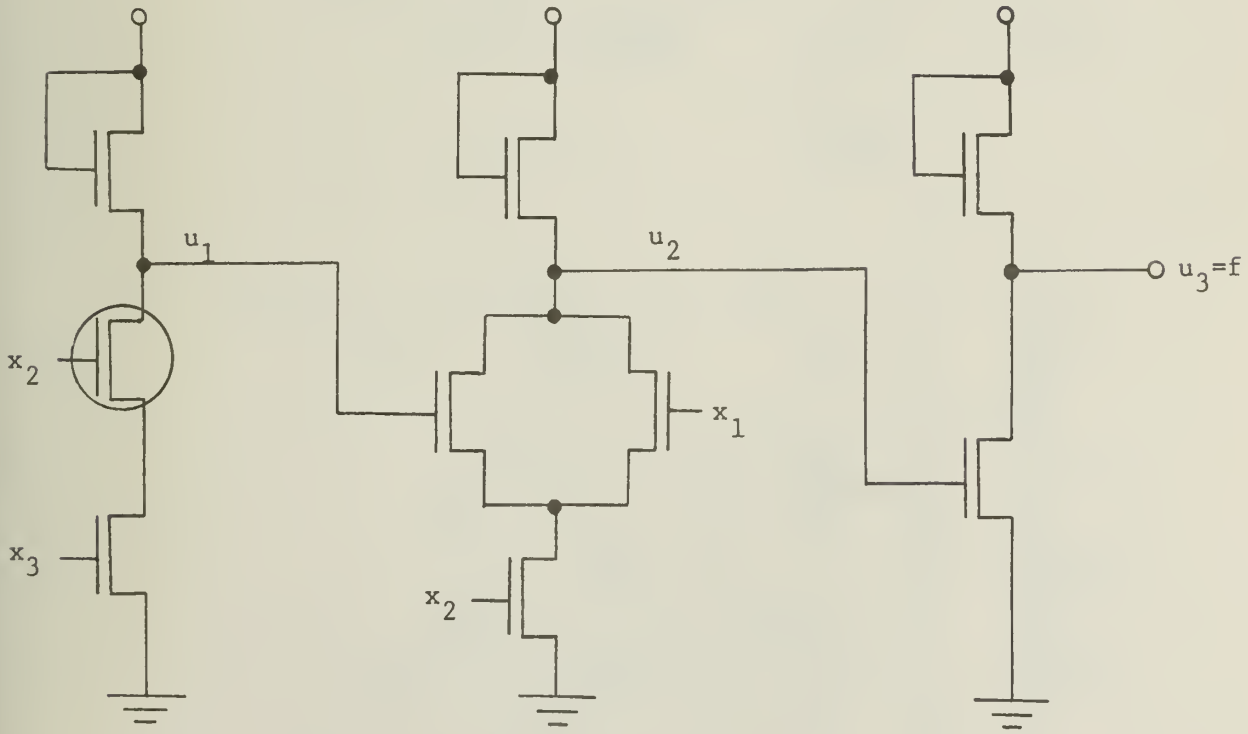


$u_1 = \overline{x_2 x_3}$

$u_2 = \overline{(u_1 \vee x_1)x_2} =$

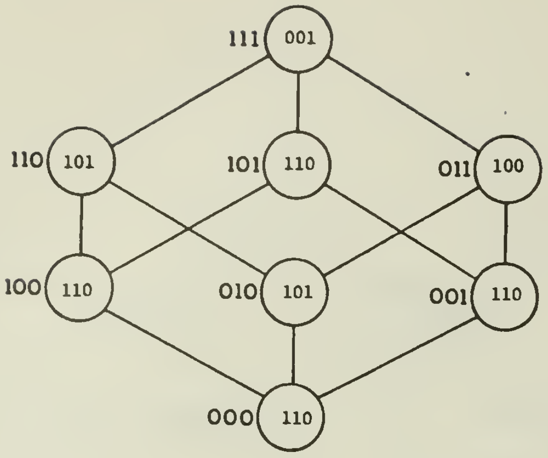
$\bar{x}_2 \vee \bar{x}_1 x_2 x_3$

$u_3 = \bar{u}_2$



(c) Label 3-cube and MOS network for NFS$(3,f)_3$.

Fig. 6.1.1 (Continued)

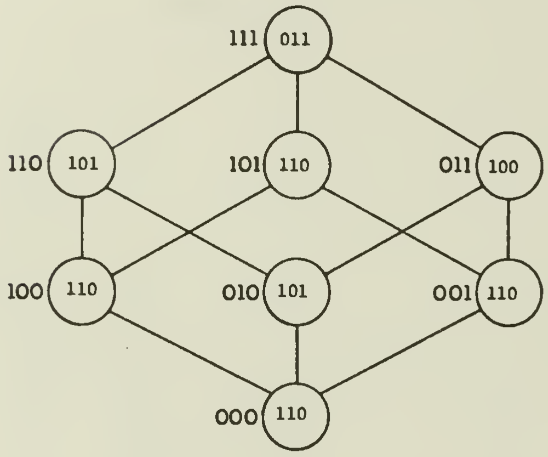


$u_1 = \overline{x_1x_2x_3}$

$u_2 = \bar{x}_2$

$u_3 = \overline{u_2 \vee u_1x_3} = x_2(x_1x_2x_3 \vee \bar{x}_3)$

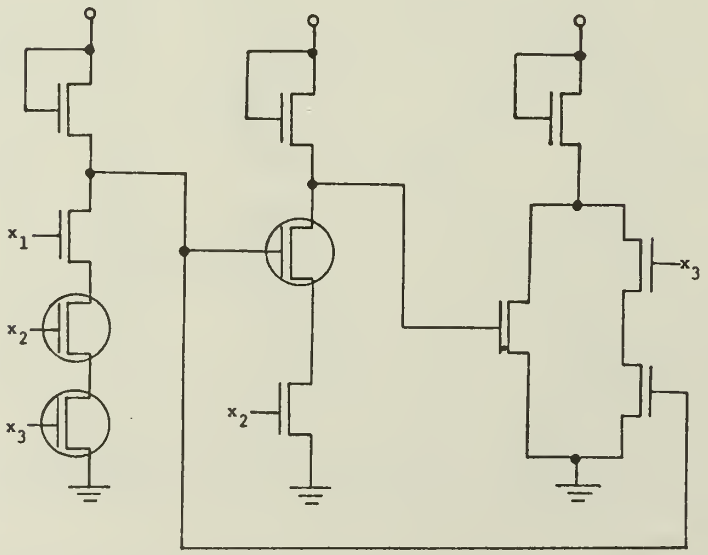
(d) Labeled 3-cube for NFS(3,f)$_4$. (An MOS network for NFS(3,f)$_4$ can be obtained from the network shown in (b) by interchanging u_1 and u_2.)



$u_1 = \overline{x_1x_2x_3}$

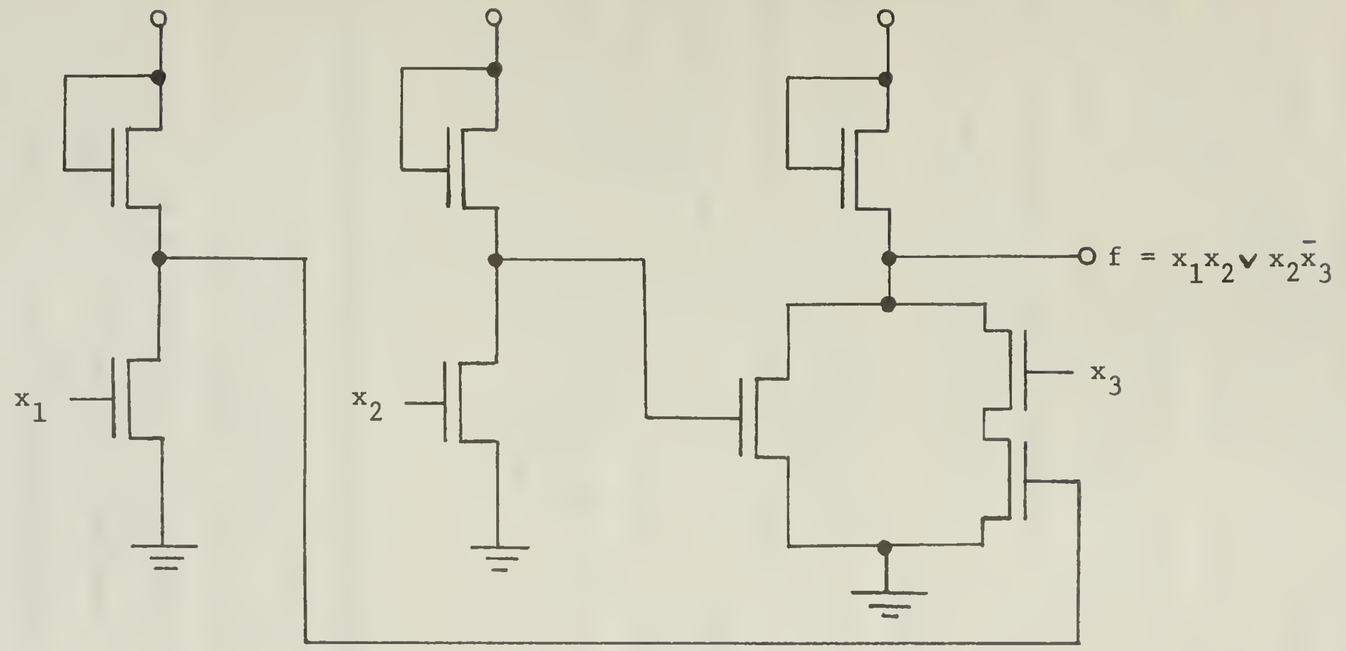
$u_2 = \overline{u_1x_2} = x_1x_2x_3 \vee \bar{x}_2$

$u_3 = \overline{u_2 \vee u_1x_3}$



(e) Labeled 3-cube and MOS network for NFS(3,f)$_5$.

Fig. 6.1.1 (Continued)



(f) An irredundant MOS network for f.

Fig. 6.1.1 (Continued)

lines in the labeled 3-cube. Figure 6.1.1(a) also shows an MOS network corresponding to $NFS(3,f)_1$: the i-th MOS cell realizes the i-th function in $NFS(3,f)_1$, for i = 1, 2, and 3. Although each MOS cell is irredundant with respect to the function of $NFS(3,f)_1$, which it realizes, the MOS network as a whole has some redundant FETs. The two encircled FETs indicate redundant FETs: the s-extraction of the FET encircled in the solid line and the o-extraction of the FET encircled in the broken line will yield an MOS network shown in Fig. 6.1.1(f) which also realizes f. All other MOS networks based on the stratified structure of f in Fig. 6.1.1 are also redundant as shown by the encircled FETs in the networks.

Example 6.1.2 - An MOS network derived from the NFS(3,f) by algorithm MNL for function $f = x_1x_2 \vee x_2\bar{x}_3$ (Example 3.1) is shown in Fig. 6.1.2. The MOS network is redundant as indicated by the encircled FETs.

The above two examples demonstrate that the design procedure of networks with a minimum number of MOS cells given by [NTK 72] and [Liu 72] are unable to design irredundant MOS networks for some functions.

6.2 Outline of the Design Algorithm of Irredundant MOS Networks

Section 6.2 presents the outline of a procedure for the synthesis of irredundant MOS networks with a minimum number of MOS cells for a single function. This procedure also consists of two phases of design, i.e., the design of the functions realized by each negative gate (MOS cell) and the design of an irredundant MOS cell for each of these functions. However, in order to guarantee the irredundancy of the

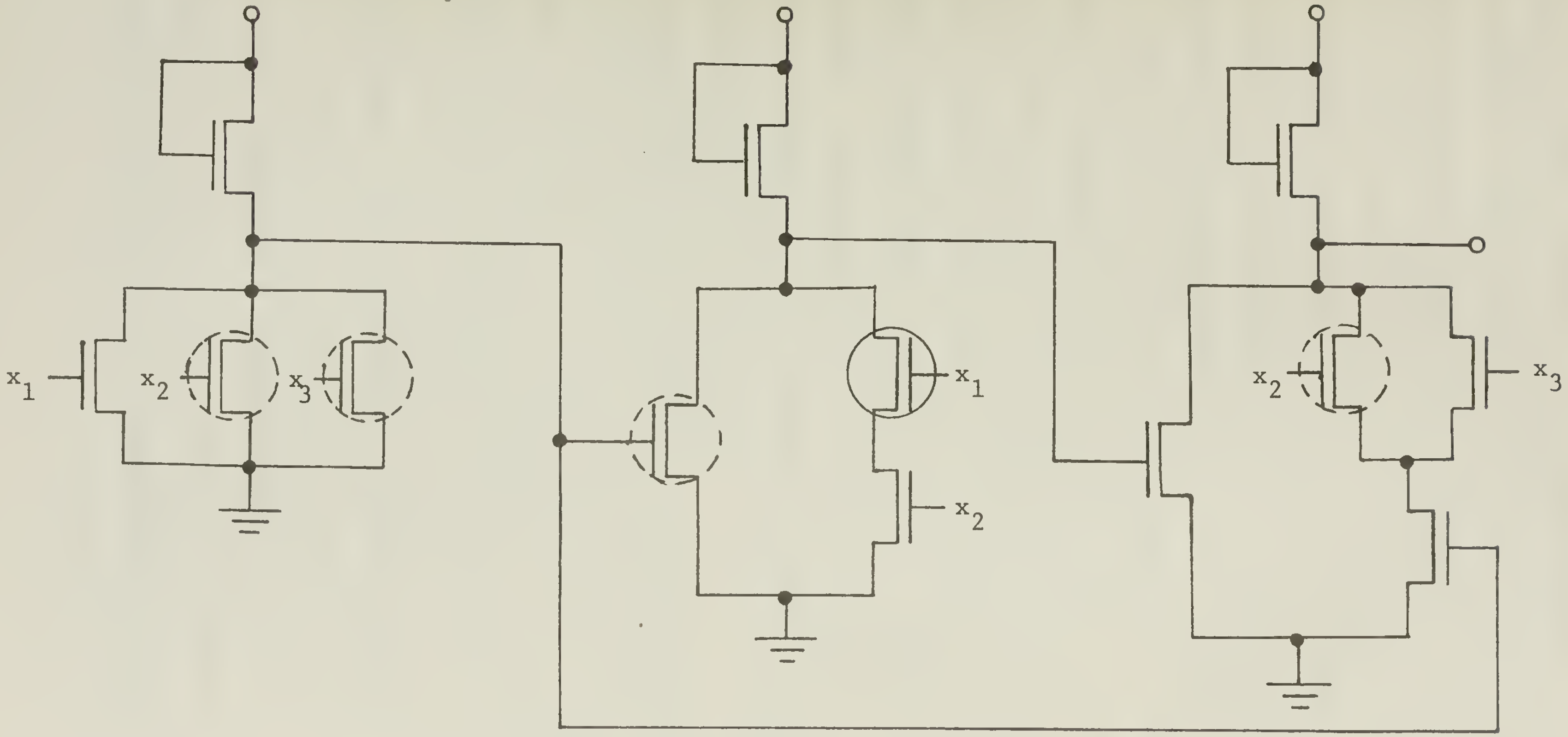


Fig. 6.1.2 Example 6.1.2: MOS network derived from the NFS(3,f) by algorithm MNL for function $f = x_1x_2 \vee x_2x_3$.

designed network, the two phases are not performed separately, unlike the case of [NTK 72] and [Liu 72], but interactively. In other words, the network will not be designed by single-pass application of Phase I followed by Phase II, but iterations of application of Phase I followed by Phase II. The motivation is to allow the function realized by the gate in the remotest level from the network output be specified by Phase I as flexible as possible (containing as many unspecified components as possible. The formal definition will be given later). Phase II can then design the internal structure of an irredundant MOS cell for this function which is usually incompletely specified with respect to the set of external variables, $x_1,\ldots, x_n$. Only when Phase II for this gate has been completed, does this function become completely specified with respect to the set of external variables, $x_1,\ldots,x_n$. This process will be repeatedly applied to the gate in the next remotest level until the network output is reached. This design procedure will guarantee that the designed MOS network is irredundant because each function will be specified to contain as many don't care components as possible. By this we mean that each specified component (i.e., not a don't care) for the function for each gate is absolutely required for the realization of the network output, and therefore, any change in these specified components caused by extraction of FETs from this cell will change the network output.

In order to explain the irredundancy of the designed network, let us consider an MOS network designed by this approach. Since each MOS cell g_i in this network is irredundant with respect to the function that has been specified so as to contain as many don't cares as possible, any extraction of FETs from this cell g_i will cause some change in specified values, say $u_i(A)$, of its corresponding function u_i, which in turn will definitely lead to changes in the output function of the network. Otherwise, those changed components of the corresponding function, $u_i(A)$, would have been specified as don't cares during the design process. This means that the designed MOS network is irredundant. A formal proof will be given after the algorithm is introduced.

The above is the motivation for the procedure to be introduced in Section 6.4. The outline of the procedure is given below.

Let us consider the problem of synthesizing an irredundant MOS network with a minimum number of MOS cells for a completely specified function of n variables, $f(x_1,\ldots,x_n)$. Let $NFS(R_f,f) = (u_1,\ldots,u_{R_f-1}, u_{R_f}=f)$ be the minimum negative function sequence corresponding to the irredundant MOS network to be designed.

<u>Outline of the design algorithm of irredundant MOS networks</u>.

Given a function f, R_f is the minimum number of negative gates (MOS cells) required for the realization of f. At the start of this procedure, the entire NFS except u_{R_f} is unspecified, i.e., $NFS^o(R_f,f) = \{u_1^*,\ldots,u_{R_f-1}^*,f\}$ where superscript o means that no function except $u_{R_f} = f$ is specified and u_i^* means the unspecified function for u_i, i.e., all components of u_i^* are don't cares.

<u>Step 0</u> - Set i=1.

<u>Step 1</u> - Specify $\tilde{u}_i$ in such a manner that $\tilde{u}_i$ contains as many don't care components as possible which is negative with respect to

$x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$. (This step is entered with a partially specified negative function sequence $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},u_i^*,\ldots,u^*_{R_f-1},f)$, and will yield $\tilde{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\tilde{u}_i,u^*_{i+1},\ldots,u^*_{R_f-1},f)$.)

<u>Step 2</u> - Obtain an irredundant MOS cell configuration for the incompletely specified function $\tilde{u}_i$ with inputs chosen from $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$. (Algorithms 5.1 and 5.2 may be used). Let u_i be the function realized by this MOS cell which is now completely specified with respect to $x_1,\ldots,x_n$. (After this step $NFS^i(R_f,f) = (u_1,\ldots,u_i,u^*_{i+1},\ldots,u^*_{R_f-1},f)$ is obtained.)

<u>Step 3</u> - If $i=R_f-1$, design an irredundant MOS cell configuration for f with possible inputs from $x_1,\ldots,x_n,u_1,\ldots,u_{R_f-1}$, and terminate this procedure; otherwise set $i=i+1$, and go to Step 2.

The nucleus of this algorithm is Steps 1 and 2. Step 2 designs an irredundant MOS cell configuration for a given incompletely specified function which has been discussed in Section 5, so we will, in Section 6.3, concentrate on Step 1, i.e., specifying $\tilde{u}_i$ based on a partially specified negative function sequence, $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},u_i^*,\ldots,u^*_{R_f-1},f)$ so as to contain as many don't cares as possible.

6.3 <u>Maximum Permissible Function</u>

The following definitions formally define partially specified negative function sequences and $\tilde{u}_i$ mentioned in Section 6.2.

Definition 6.3.1 - A partially[†] specified negative function sequence of length R_f and degree i for a function f is a sequence of R_f functions denoted by $NFS^i(R_f,f) = (u_1,\ldots,u_i,u^*_{i+1},\ldots,u^*_{R_f-1},u_{R_f})$ such that:

(i) $u_1,\ldots,u_i$ are completely specified functions of $x_1,\ldots,x_n$;

(ii) $(u_1,\ldots,u_i)$ is an NFS of length i;

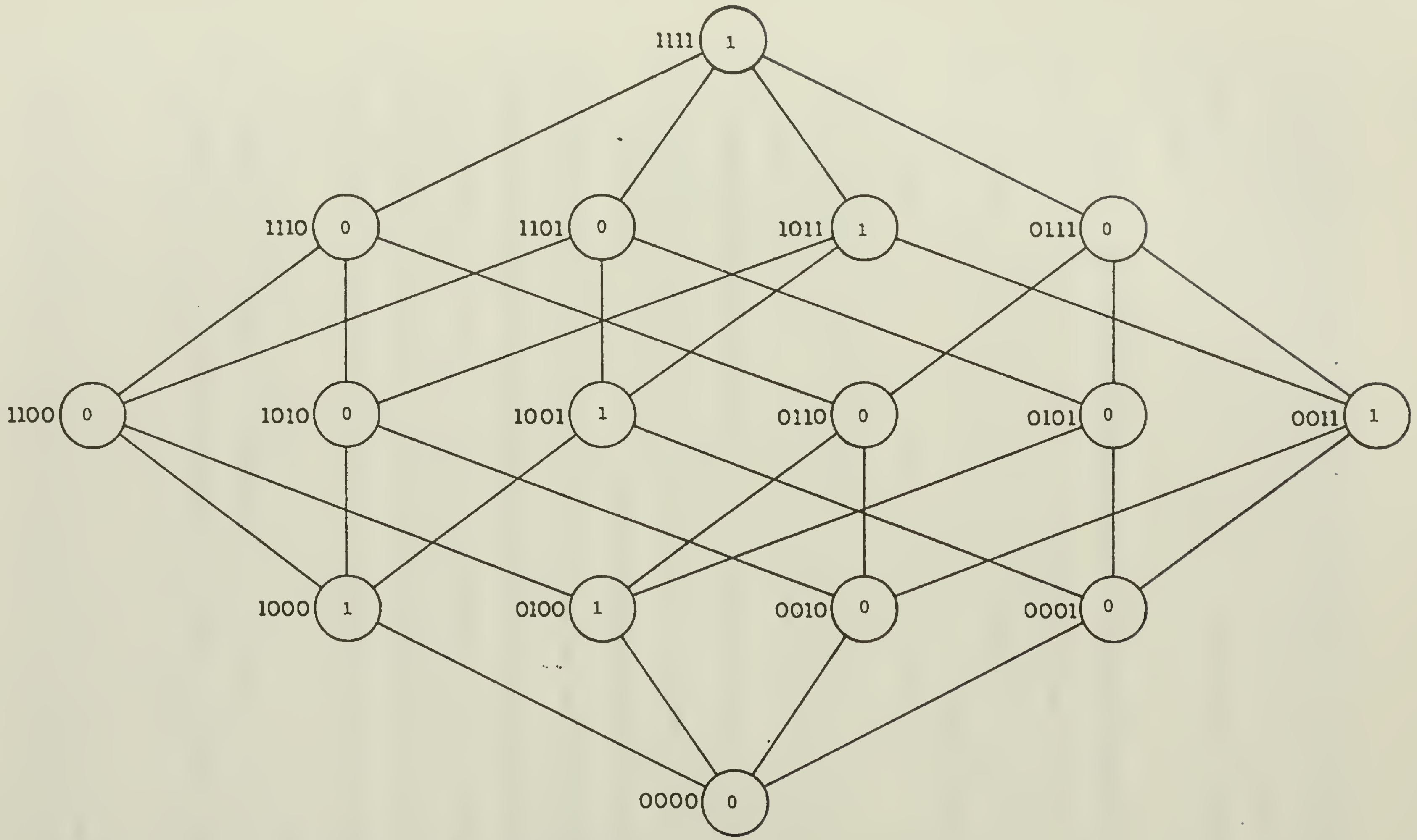
(iii) $u^*_{i+1},\ldots,u^*_{R_f-1}$ are unspecified functions (i.e., all components are don't cares); and

(iv) $u_{R_f}=f$.

A partially specified $NFS^i(R_f,f)$ is feasible if there exists at least one complete specification (with respect to $x_1,\ldots,x_n$) $u_{i+1},\ldots,u_{R_f-1}$ of $u^*_{i+1},\ldots,u^*_{R_f-1}$, such that $(u_1,\ldots,u_{R_f-1},f)$ which will be referred to as a completion of $NFS^i(R_f,f)$ is an $NFS(R_f,f)$; otherwise $NFS^i(R_f,f)$ is an infeasible partially specified NFS.

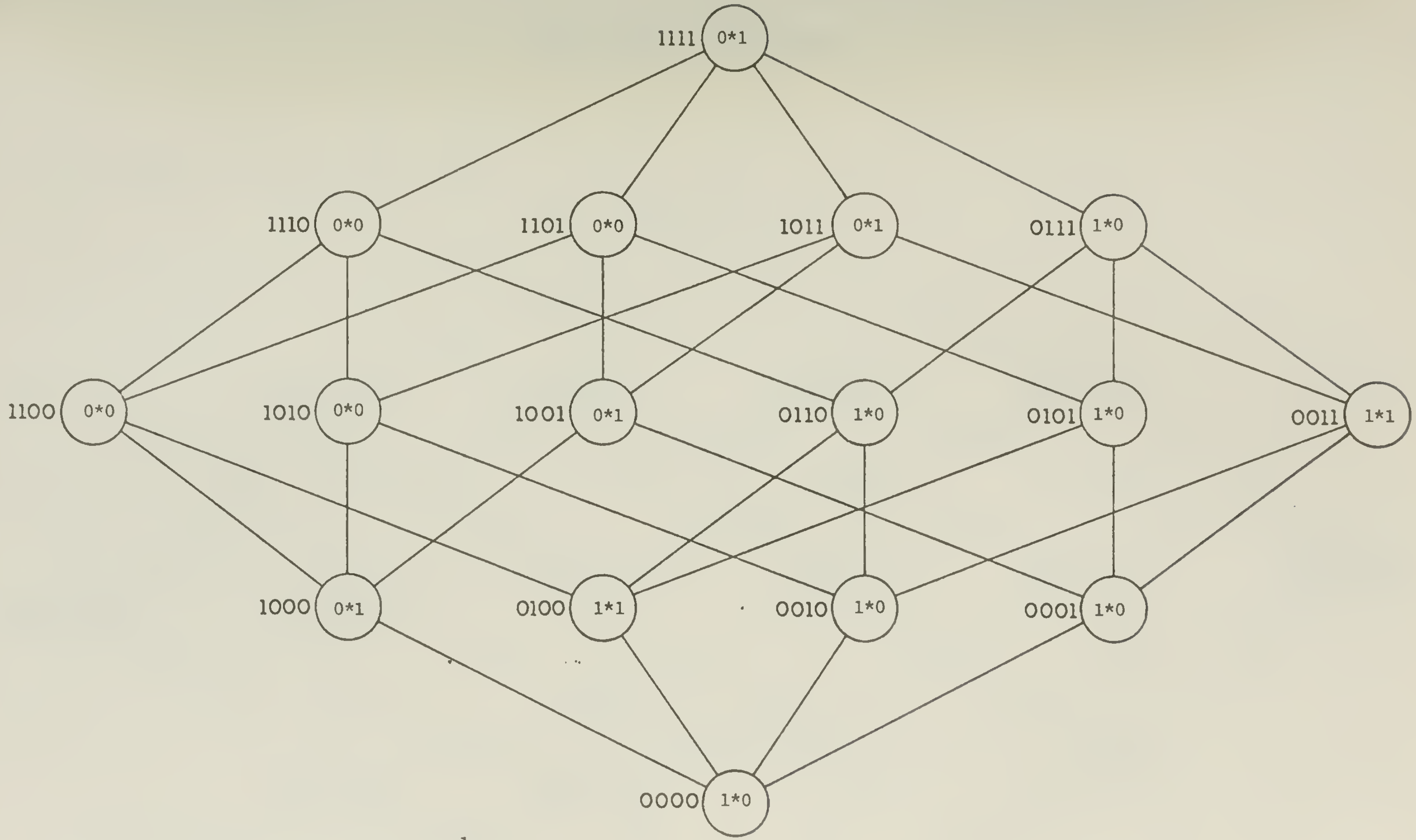
Example 6.3.1 - Let us consider a function f of four variables as shown in Fig. 6.3.1(a). We will show some partially specified NFS for f. Fig. 6.3.1(b) shows a feasible partially specified NFS of length 3 and degree 1 for f, $NFS^1(3,f)$, where u_1 is specified as $u_1 = \bar{x}_1$. This is a feasible partially specified NFS since a completion of this partial specification shown in Fig. 6.3.1(c) results in an NFS(3,f). On the other hand, another partially specified NFS of length 3 and degree 1 for f, $(u_1 = \bar{x}_2,\ u^*_2,\ f)$, shown in Fig. 6.3.1(d) is an infeasible $NFS^1(3,f)$;

[†]Although the word "partially specified" has a meaning similar to "incompletely specified," the latter is used only in conjunction with a single function, in order to avoid confusion.



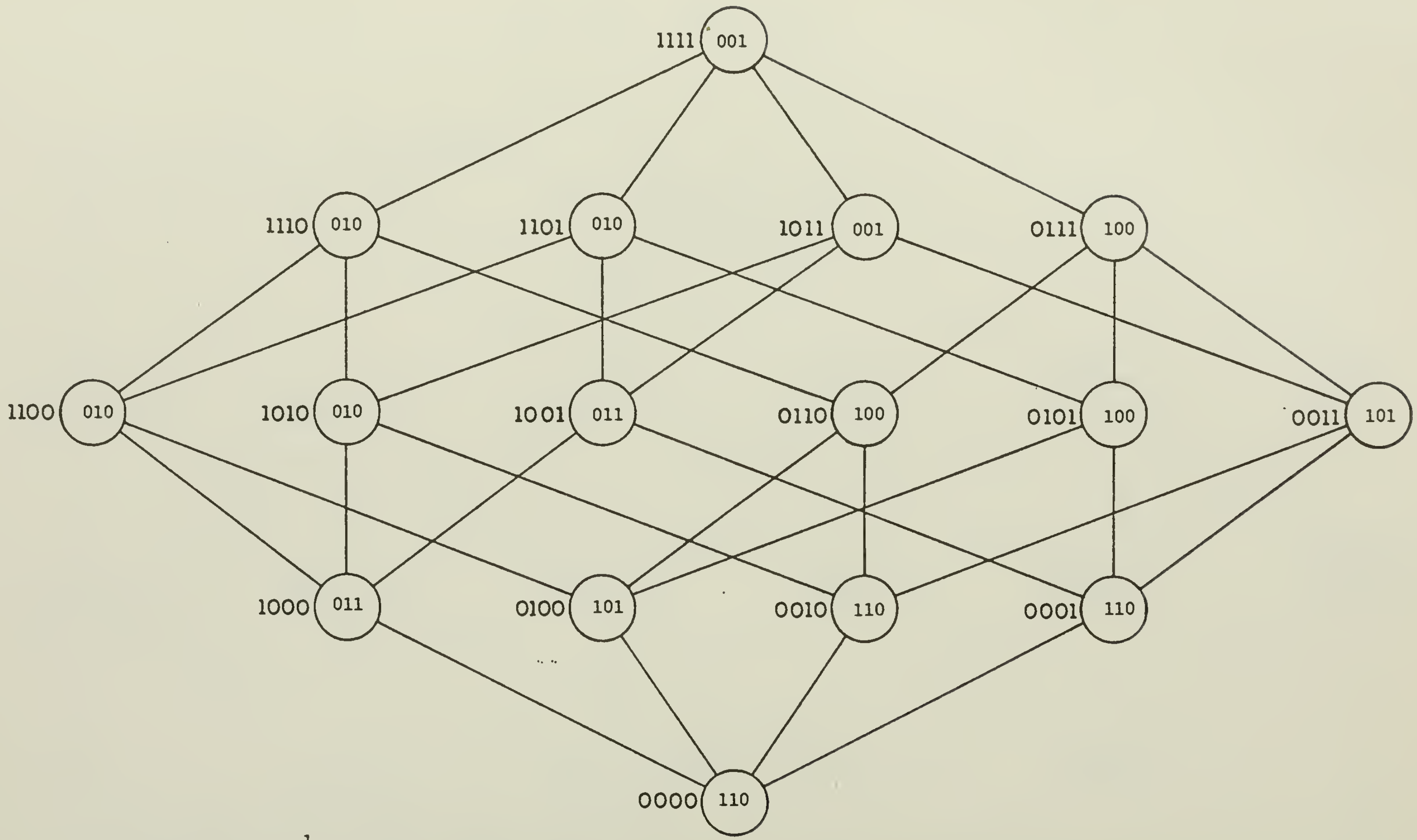
(a) A completely specified function f of 4 variables.

Fig. 6.3.1 Example 6.3.1.



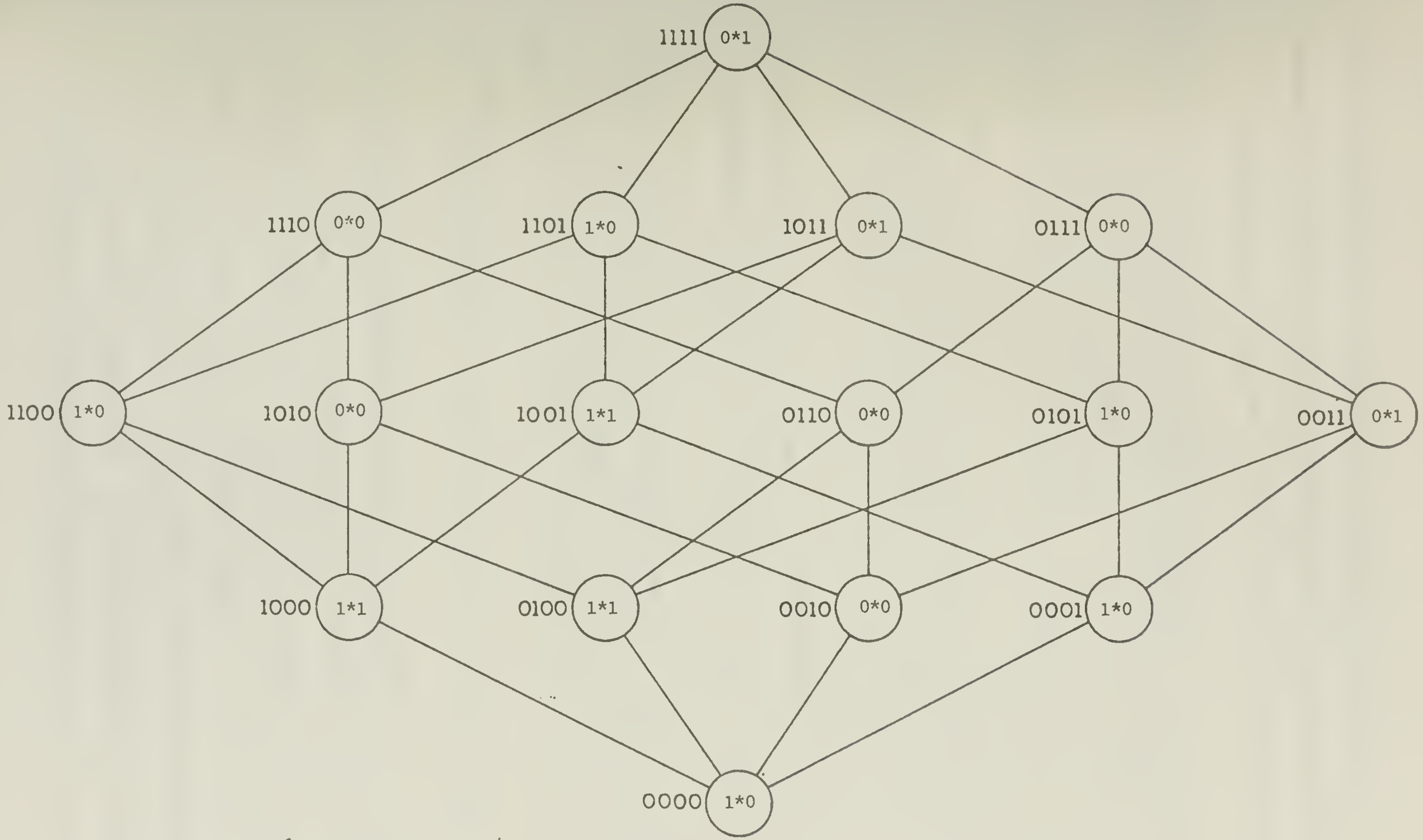
(b) A partially specified feasible $NFS^1(3,f)$.

Fig. 6.3.1 (Continued)



(c) A completion of $NFS^1(3,f)$ shown in (b).

<u>Fig. 6.3.1</u> (Continued)



(d) An infeasible $NFS^1(3,f) = (x_1 = \bar{x}_2, u_2^*, f)$.

<u>Fig. 6.3.1</u> (Continued)

since no matter how u_2 is specified, one of the two inverse edges will remain along the critical path (1111)→(0111)→(0011)→(0010) (two inverse edges along the path but only one bit can be used to cover them).

<u>Definition 6.3.2</u> - <u>The maximum permissible function</u> $\tilde{u}_i$ for a feasible partially specified NFS of length R_f and degree i-1 for function f, $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1}, u_i^*,\ldots,u_{R_f-1}^*,f)$, is an incompletely specified function for u_i^* such that:

(i) $\tilde{u}_i$ is negative with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$;

(ii) every negative completion u_i of $\tilde{u}_i$ yields a feasible $NFS^i(R_f,f) = (u_1,\ldots,u_i,u_{i+1}^*,\ldots,u_{R_f-1}^*,f)$; and

(iii) any function u_i' which is not a completion of u_i will make corresponding $NFS'^i(R_f,f) = (u_1,\ldots,u_{i-1},u_i',u_{i+1}^*,\ldots,u_{R_f-1}^*,f)$ infeasible.

It should be noted that the condition (iii) in the above definition is necessary since we want to have a <u>maximum</u> permissible function that contains as many don't care components as possible. In other words, each specified value of u_i is absolutely required, as expressed by condition (iii).

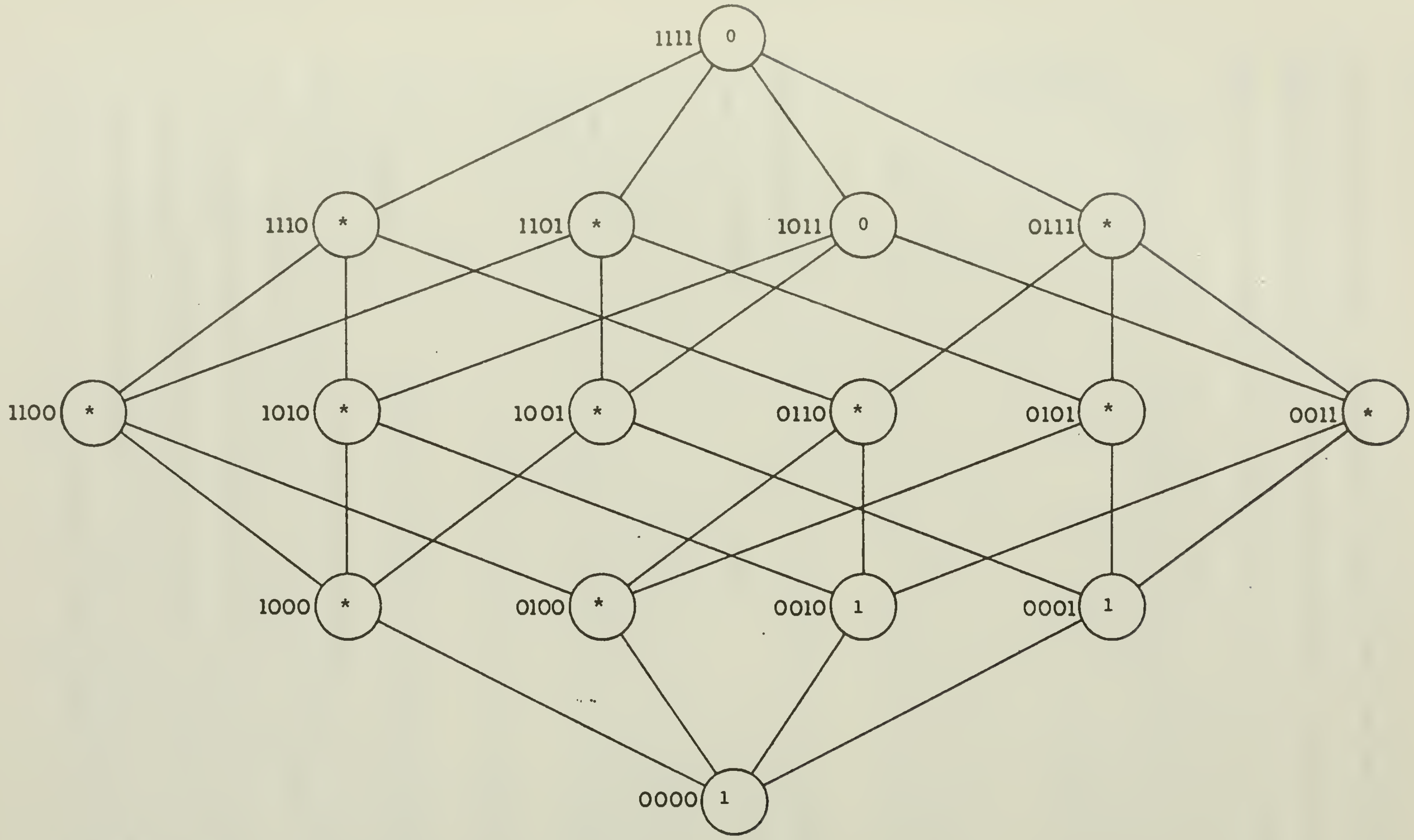
<u>Example 6.3.2</u> - Consider the function f discussed in Example 6.3.1. The maximum permissible function $\tilde{u}_1$ for $NFS^o(R_f,f)$ is shown in Fig. 6.3.2(a) where $\tilde{u}_1(1111) = 0$, $\tilde{u}_1(1011) = 0$, $\tilde{u}_1(0010) = 1$, $\tilde{u}_1(0001)=1$, and $\tilde{u}_1(0000)=1$ are the only specified values of $\tilde{u}_1$. If we change any one of these specified values to its complement, the resulting $(\tilde{u}_1', u_2^*, f)$ will no longer be a feasible $NFS^1(3,f)$. For example, if we set $\tilde{u}_1'(1011)=1$, we must set $\tilde{u}_1'(1010)=\tilde{u}_1'(1000)=\tilde{u}_1'(0000)=1$ as shown in

Fig. 6.3.2(b) (underlined) because $\tilde{u}_1'$ must be a negative function with respect to $x_1,\dots,x_n$. Then no matter how $\tilde{u}_2'$ is specified, there will be an inverse edge along the path (1011)-(1010)-(1000)-(0000) in the resulting labeled 4-cube. (There are two inverse edges along this path in $C_4(f)$, but only one bit (u_2) is allowed to make the labeled 4-cube $C_4(f)$ not have inverse edges.) It can be proved similarly that any of the other four specified values of $\tilde{u}_1$ cannot be changed without causing the resulting $NFS^1(3,f)$ infeasible. On the other hand, every negative completion u_1 of $\tilde{u}_1$ makes $(u_1,\overset{*}{u}_2,f)$ a feasible $NFS^1(3,f)$. This can be proved exhaustively, but it is omitted here.

Now we are ready to present an algorithm which obtains $\tilde{u}_i$ based on a given $NFS^{i-1}(R_f,f) = (u_1,\dots,u_{i-1},\overset{*}{u}_i,\dots,\overset{*}{u}_{R_f-1},f)$. This algorithm involves two major steps, one of which is an algorithm to obtain conditional minimum labeling (CMNL) and the other conditional maximum labeling (CMXL) which are similar to algorithms MNL and MXL, respectively.

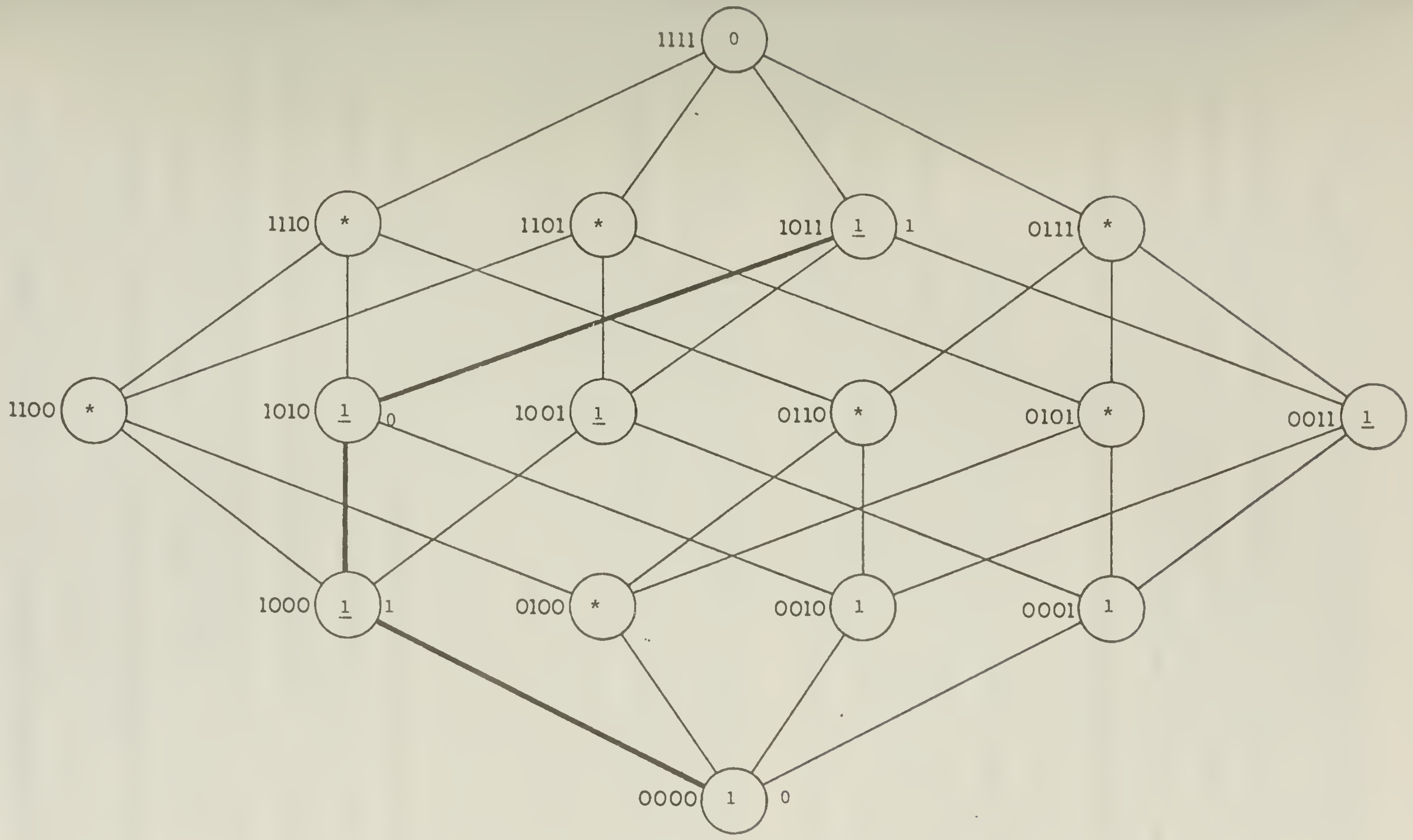
<u>Algorithm 6.3.1</u> - Conditional Minimum Labeling (<u>CMNL</u>)

This algorithm obtains a conditional minimum labeling for a given feasible $NFS^{i-1}(R_f,f) = (u_1,\dots,u_{i-1},\overset{*}{u}_i,\dots,\overset{*}{u}_{R_f-1},f)$. The conditional minimum labeling of $NFS^{i-1}(R_f,f)$ is a completion $\underline{NFS}^{i-1}(R_f,f) = (u_1,\dots, u_{i-1},\underline{u}_i,\dots,\underline{u}_{R_f-1},f)$ of $NFS^{i-1}(R_f,f)$ such that the label $\ell(A;u_1,\dots,u_{i-1}, \underline{u}_i,\dots, \underline{u}_{R_f-1},f) \equiv L_{mn}^{f,i-1}(A)$ for each A in the corresponding labeled n-cube $C_n(u_1,\dots,u_{i-1},\underline{u}_i,\dots,\underline{u}_{R_f-1},f)$ takes the minimum possible value among all feasible completions of $NFS^{i-1}(R_f,f)$.



(a) $\tilde{u}_1$ for the f of Fig. 6.3.1(a) (Example 6.3.1).

<u>Fig. 6.3.2</u> Example 6.3.2.



(b) Changing a specified value $\tilde{u}_1(1011)=0$ to $\tilde{u}_1'(1011)=1$ makes $(\tilde{u}_1', u_2^*, f)$ infeasible. The bold lines show a critical path along which an inverse edge will exist no matter how u_2^* is assigned.

Fig. 6.3.2 (Continued)

Step 1 - Assign as $L_{mn}^{f,i-1}(I)$ the value $\sum_{k=1}^{i-1} 2^{R_f-k} u_k(I) + f(I)$.

Step 2 - When $L_{mn}^{f,i-1}(A)$ is assigned to each vertex A of weight w ($1 \leq w \leq n$) of C_n, assign as $L_{mn}^{f,i-1}(B)$ to each vertex B of weight w-1 the smallest binary integer satisfying the following three conditions:

(a) The k-th most significant bit of $L_{mn}^{f,i-1}(B)$ is $u_k(B)$ for $k=1,\ldots,i-1$;

(b) The least significant bit of $L_{mn}^{f,i-1}(B)$ is f(B):

(c) $L_{mn}^{f,i-1}(B) \geq L_{mn}^{f,i-1}(A)$ for every directed edge $\vec{AB}$ terminating at B. (Notice that each of the i-th through (R_f-1)-st bits of $L_{mn}^{f,i-1}(B)$ is 0 or 1 depending on what $L_{mn}^{f,i-1}(B)$ is found as the smallest binary integer.)

Step 3 - Repeat Step 2 until $L_{mn}^{f,i-1}(0)$ is assigned.

Step 4 - The k-th most significant bit of $L_{mn}^{f,i-1}(A)$ is denoted by $\underline{u}_k(A)$ for $k=i,\ldots,R_f-1$ and for each $A \in C_n$, $\underline{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1}, \underline{u}_i,\ldots,\underline{u}_{R_f-1},f)$ has been obtained as the completion of $NFS^{i-1}(R_f,f)$ by CMNL.

The validity of this algorithm is obvious. Since $NFS^{i-1}(R_f,f)$ is a feasible partially specified NFS, it must have a feasible completion. The algorithm assigns a minimum possible value as a label to each vertex, and conditions (A) and (B) of Step 2 guarantee that the resulting labeled n-cube is a completion of $NFS^{i-1}(R_f,f)$ while condition (c) of Step 2 guarantees that the labeled n-cube has no inverse edge, constituting an NFS.

Similarly, algorithm MXL can be modified to obtain a conditional maximum completion of $NFS^{i-1}(R_f,f)$.

<u>Algorithm 6.3.2</u> - Conditional Maximum Labeling (<u>CMXL</u>)

This algorithm obtains a conditional maximum labeling based on a given feasible $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},u_i^*,\ldots,u_{R_f-1}^*,f)$ for a function f. The conditional maximum labeling of $NFS^{i-1}(R_f,f)$ is a complete specification $\hat{NFS}^{i-1}(R_f,f)=(u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_{R_f-1},f)$ of $NFS^{i-1}(R_f,f)$ such that the label $\ell(A;\ u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_{R_f-1},f) \equiv L_{mx}^{f,i-1}(A)$ for each A in the corresponding labeled n-cube $C_n(u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_{R_f-1},f)$ takes the maximum possible value among all feasible completions of $NFS^{i-1}(R_f,f)$.

<u>Step 1</u> - Assign as $L_{mx}^{f,i-1}(0)$ the value $\sum_{k=1}^{i-1} 2^{R_f-k} u_k(0) + \sum_{k=i}^{R_f-1} 2^{R_f-k} + f(0)$.

<u>Step 2</u> - When $L_{mx}^{f,i-1}(A)$ is assigned to each vertex A of weight $w(o \leq w \leq n-1)$ of C_n, assign as $L_{mx}^{f,i-1}(B)$ to each B of weight w+1 the largest binary integer satisfying the following three conditions:

(a) The k-th most significant bit of $L_{mx}^{f,i-1}(B)$ is $u_k(B)$ for $k=1,\ldots,i-1$;

(b) The least significant bit of $L_{mx}^{f,i-1}(B)$ is f(B);

(c) $L_{mx}^{f,i-1}(B) \leq L_{mx}^{f,i-1}(A)$ for every directed edge $\vec{BA}$ originating from B. (Notice that each of the i-th through (R_f-1)-st bits of $L_{mx}^{f,i-1}(B)$ is 0 or 1 depending on what $L_{mx}^{f,i-1}(B)$ is found as the largest binary integer.)

<u>Step 3</u> - Repeat Step 2 until $L_{mx}^{f,i-1}(I)$ is assigned.

<u>Step 4</u> - The k-th most significant bit of $L_{mx}^{f,i-1}(A)$ is denoted by $\hat{u}_k(A)$ for $k=i,\ldots,R_f-1$ and for each $A \varepsilon C_n$, $\hat{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\hat{u}_i,\ldots,\hat{u}_{R_f-1},f)$ has been obtained as the completion of $NFS^{i-1}(R_f,f)$ by CMXL.

The validity of this algorithm is obvious since $NFS^{i-1}(R_f,f)$ is a feasible partially specified NFS which has at least one $NFS(R_f,f)$ completion. This algorithm assigns the maximum possible label to each vertex of the labeled n-cube, and conditions in Step 2 guarantee the labeled n-cube is an $NFS(R_f,f)$.

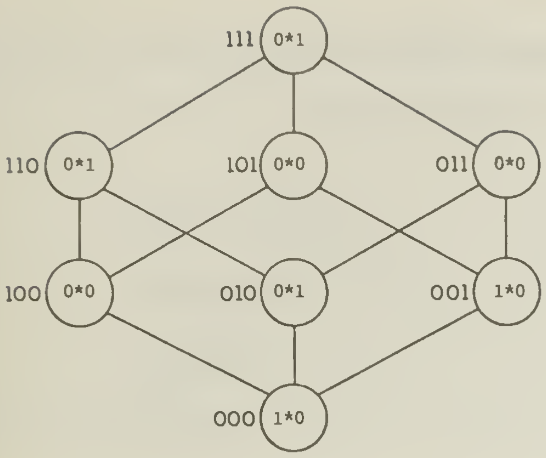
Example 6.3.3 - Consider the function discussed in Examples 3.1 and 3.4. The function f is shown in Fig. 3.2(a) (also in Fig. 3.3 (a)), and the $C_3(u_1,u_2,f)$s based on MNL and MXL are shown in Fig. 3.2(b) and Fig. 3.3(b), respectively. These two labeled 3-cubes can also be obtained by CMNL and CMXL for $NFS^o(R_f,f) = (u_1^*,u_2^*,f)$.

Now let us consider an $NFS^1(R_f,f) = (u_1,u_2^*,f)$ shown in Fig. 6.3.3 (a) in the form of a partially labeled 3-cube. According to algorithm CMNL, $NFS^1(3,f)$ is completed as $\underline{NFS}^1(3,f)$ as shown in Fig. 6.3.3(b). Similarly, $NFS^1(3,f)$ is completed by algorithm CMXL as $\hat{NFS}^1(3,f)$ as shown in Fig. 6.3.3(c).

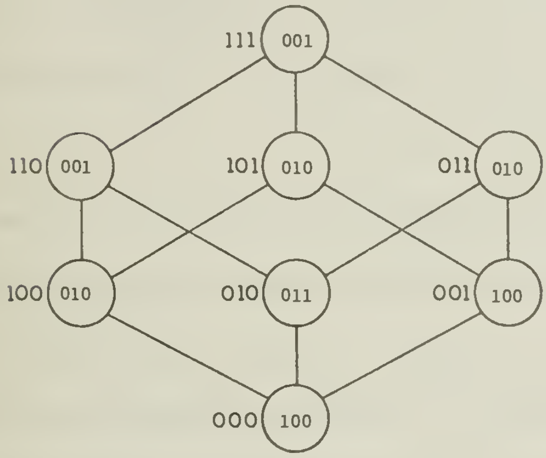
The following algorithm obtains the maximum permissible function $\tilde{u}_i$ for a feasible partially specified $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},u_i^*,\ldots,u_{R_f-1}^*,f)$. The proof of this algorithm also shows the existence of the maximum permissible function defined by Definition 6.3.2.

Algorithm 6.3.3 - Derivation of the maximum permissible function $\tilde{u}_i$ for a given $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},u_i^*,\ldots,u_{R_f-1}^*,f)$ (MPF)

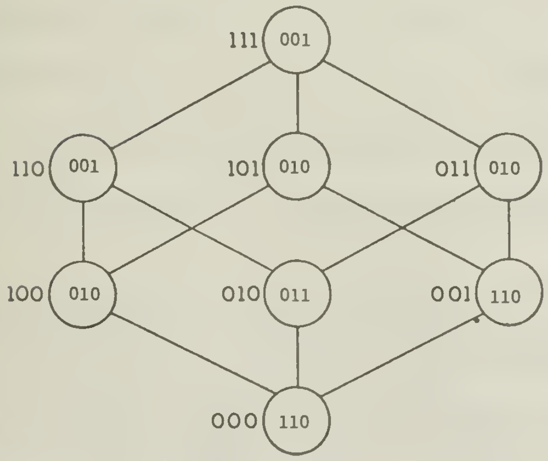
Step 1 - Obtain a completion $\underline{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\underline{u}_i,\underline{u}_{i+1},\ldots,\underline{u}_{R_f-1},f)$ of $NFS^{i-1}(R_f,f)$ according to algorithm CMNL.



(a) $NFS^1(3,f) = (u_1 = \overline{x_1 \vee x_2}, u_2^*, u_3 = f)$.



(b) $\underline{NFS}^1(3,f) = (u_1, u_2, f)$ obtained from $NFS^1(3,f)$ by algorithm CMNL.



(c) $\widehat{NFS}^1(3,f) = (u_1, \hat{u}_2, f)$ obtained from $NFS^1(3,f)$ by algorithm CMXL.

Fig. 6.3.3 Example 6.3.3.

Step 2 - Obtain another completion $\widehat{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},\hat{u}_i,\hat{u}_{i+1},\ldots,\hat{u}_{R_f-1},f)$ according to algorithm CMXL.

Step 3 - For each vertex A of C_n, do the following:

(i) assign $\tilde{u}_i(A)=0$ if and only if $\underline{u}_i(A)=0$ and $\hat{u}_i(A)=0$,

(ii) assign $\tilde{u}_i(A)=1$ if and only if $\underline{u}_i(A)=1$ and $\hat{u}_i(A)=1$,

(iii) assign $\tilde{u}_i(A)=*$ (unspecified) if and only if $\underline{u}_i(A)=0$ and $\hat{u}_i(A)=1$

Theorem 6.3.1 - The function obtained by algorithm MPF is the maximum permissible function for the feasible partially specified $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1},u^*_i,\ldots,u^*_{R_f-1},f)$.

Proof - To prove $\tilde{u}_i$ is the maximum permissible function for $NFS^{i-1}(R_f,f)$, we have to prove: (i) there is always a solution $\tilde{u}_i$ which is negative with respect to $x_1,\ldots,x_n,u_1,\ldots,u_{i-1}$ if $NFS^{i-1}(R_f,f)$ is a feasible partially specified NFS, (ii) changing any specified component of $\tilde{u}_i$, i.e., changing $\tilde{u}_i(A)=0$ to $\tilde{u}'_i(A)=1$ or $\tilde{u}_i(A)=1$ to $\tilde{u}'_i(A)=0$ will make the corresponding $\widetilde{NFS}^{i-1}(R_f,f)$ infeasible, and (iii) any negative completion u_i of $\tilde{u}_i$ (with respect to $x_1,\ldots,x_n,u_1,\ldots,u_{i-1}$) will make the corresponding $(u_1,\ldots,u_i,u^*_{i+1},\ldots,u^*_{R_f-1},f)$ to be a feasible partially specified NFS of length R_f and degree i for function f.

(i) Since the given partially specified NFS is feasible, there must exist at least one completion $(u_1,\ldots,u_{i-1},u_i,\ldots,u_{R_f-1},f)$ of $(u_1,\ldots,u_{i-1},u^*_i,\ldots,u^*_{R_f-1},f)$ such that $C_n(u_1,\ldots,u_{R_f-1},f)$ is a labeled n-cube without any inverse edge. Algorithm CMNL assigns the minimum possible value to each vertex of C_n so that the resulting labeled n-cube has no inverse edge. Therefore Step 1 of algorithm MPF can always be successfully executed without any problem (e.g., the conditions in Step 2 of CMNL may not be simultaneously satisfied without additional bits). Similarly, Step 2 of algorithm MPF can be successfully executed without any problem because algorithm CMXL can be successfully executed. Besides, for every vertex A in C_n, $L_{mn}^{f,i-1}(A) \leq L_{mx}^{f,i-1}(A)$, and therefore $\underline{u}_i(A) \leq \hat{u}_i(A)$. The three cases in Step 3 of algorithm MPF exhaust all possible combinations of $\underline{u}_i(A)$ and $\hat{u}_i(A)$, so that $\tilde{u}_i$ can always be determined by algorithm MPF for a given $NFS^{i-1}(R_f,f)$. Furthermore, there must be some vertex A such that $\tilde{u}_i(A)=0$ since otherwise (i.e., if $\tilde{u}_i=1$, or * for each vertex A) $\hat{u}_i=1$ for every A results, and $(u_1,u_2,\ldots,u_{i-1},\hat{u}_{i+1},\ldots,\hat{u}_{R_f-1},f)$ without $\hat{u}_i=1$ would be an NFS of length R_f-1 for f which contradicts the assumption of the NFS having a minimum length R_f. Similarly, there must be some vertex B such that $\tilde{u}_i(B)=1$ since otherwise $(u_1,u_2,\ldots,u_{i-1},\underline{u}_{i+1},\ldots,$ $\underline{u}_{R_f-1},f)$ would be an NFS which results in the same contradiction. This proves that algorithm MPF can always be successfully executed.

Now we have to show that $\tilde{u}_i$ is negative with respect to $x_1,\ldots,x_n,$ $u_1,\ldots,u_{i-1}$. According to the definition of $NFS^{i-1}(R_f,f)$, $(u_1,\ldots,u_{i-1})$ is an NFS of length i-1, and therefore $C_n(u_1,\ldots,u_{i-1})$ has no inverse edge. Assume $\tilde{u}_i$ is not negative with respect to $x_1,\ldots,x_n,u_1,\ldots,u_{i-1}$, then by Theorem 2.3 there must exist two vertices A and B in C_n such that (a) $(a_1,\ldots,a_n,u_1(A),\ldots,u_{i-1}(A)) > (b_1,\ldots,b_n,u_1(B),\ldots,u_{i-1}(B))$, and (b) $\tilde{u}_i(A)=1$ and $\tilde{u}_i(B)=0$. In order to satisfy condition (a), A>B must hold. This means that $u_k(A)=u_k(B)$ for k=1,...,i-1 must hold since

$(u_1,\ldots,u_{i-1})$ in an NFS$(i-1, u_{i-1})$ by Definition 6.3.2 and therefore $\ell(A; u_1,\ldots,u_{i-1}) \leq \ell(B; u_1,\ldots,u_{i-1})$ for $A > B$. However, this can never occur since $\tilde{u}_i(A)=1$ and $\tilde{u}_i(B)=0$ mean that $\hat{u}_i(A)=1$ and $\hat{u}_i(B)=0$, respectively, according to Step 3 of algorithm MPF, contradicting the fact that $\hat{u}_i(A)$ and $\hat{u}_i(B)$ are obtained by algorithm CMXL that produces a labeled n-cube $C_n(u_1,\ldots,u_{i-1}, \hat{u}_i,\ldots,\hat{u}_{R_f-1},f)$ without inverse edge. Thus u_i must be negative with respect to $x_1,\ldots,x_n, u_1,\ldots,u_{i-1}$.

(ii) Now let us prove that if any specified value of $\tilde{u}_i(A)$ is changed from $\tilde{u}_i(A)=1$ to $\tilde{u}_i'(A)=0$, or from $\tilde{u}_i(A)=0$ to $\tilde{u}_i'(A)=1$, then the resulting $(u_1,\ldots,u_{i-1}, \tilde{u}_i', u^*_{i+1},\ldots,u^*_{R_f-1}, f)$ will not be feasible, i.e., no completion of $(u_1,\ldots,u_{i-1}, \tilde{u}_i', u^*_{i+1},\ldots,u^*_{R_f-1}, f)$ can be an NFS(R_f,f). This can be proved easily based on the properties of the labels obtained by CMNL and CMXL. Obviously, $L^{f,i-1}_{mn}(A)$ is the minimum possible label for A among all possible labelings corresponding to feasible completions of NFS$^{i-1}(R_f,f)$. Similarly, $L^{f,i-1}_{mx}(A)$ is the maximum possible label for A. On the other hand, $\tilde{u}_i(A)=0$ means both $\hat{u}_i(A)=0$ and $\underline{u}_i(A)=0$ which implies

$$\sum_{k=1}^{i-1} 2^{R_f-k} u_k(A) \leq L^{f,i-1}_{mn}(A) \leq L^{f,i-1}_{mx} < \sum_{k=1}^{i-1} 2^{R_f-k} u_k(A) + 2^{R_f-i}.$$

However, if $\tilde{u}_i(A)=0$ is changed to $\tilde{u}_i'=1$, any completion of $(u_1,\ldots, u_{i-1}, \tilde{u}_i', u^*_{i+1},\ldots,u^*_{R_f-1},f)$ will assign a label $L(A) \geq \sum_{k=1}^{i-1} 2^{R_f-k} u_k(A) + 2^{R_f-i} > L^{f,i-1}_{mx}(A)$ which contradicts the property that $L^{f,i-1}_{mx}(A)$ is the maximum possible label attached to A for any feasible completion of $(u_1,\ldots,u_{i-1}, u^*_i,\ldots,u^*_{R_f-1})$. Consequently, $\tilde{u}_i(A)$ can not be changed from 0 to 1 without destroying the feasibility. Similarly, no $\tilde{u}_i(A)=1$ can be changed to $\tilde{u}_i'(A)=0$ because otherwise every completion of $(u_1,\ldots,u_{i-1}, \tilde{u}_i', u^*_{i+1},\ldots,u^*_{R_f-1}, f)$ will assign a label to A which is less than $L^{f,i-1}_{mn}(A)$, and thus make this sequence of functions infeasible. This completes the proof for (ii).

(iii) To prove that $\tilde{u}_i$ is indeed the maximum permissible function for $NFS^{i-1}(R_f,f)$, we have to show every negative completion u_i (with respect to $x_1,\ldots,x_n, u_1,\ldots,u_{i-1}$) of $\tilde{u}_i$ produces a feasible $NFS^i(R_f,f)$. In other words, we have to prove that $(u_1,\ldots,u_i, u^*_{i+1},\ldots,u^*_{R_f-1}, f)$ has at least one completion which is an $NFS(R_f,f)$. Let us consider a labeled n-cube $C_n(u_1,u_2,\ldots,u_i,v_{i+1}, \ldots,v_{R_f-1}, f)$ such that

$$v_j(A) = \underline{u}_j(A) \text{ for } j = i+1,\ldots,R_f-1 \text{ if } u_i(A)=0; \text{ and}$$

$$v_j(A) = \hat{u}_j(A) \text{ for } j = i+1,\ldots,R_f-1 \text{ if } u_i(A)=1.$$

Since u_i is a completion of $\tilde{u}_i$, $\tilde{u}_i(A)=0$ means either $\tilde{u}_i(A)=0$ (i.e., both $\underline{u}_i(A) = \hat{u}_i(A) = 0$) or $\tilde{u}_i(A) = *$ (i.e., $\underline{u}_i(A)=0$ and $\hat{u}_i(A)=1$). Similarly, $u_i(A)=1$ means either $\tilde{u}_i(A) = 1$ (i.e., both $\underline{u}_i(A)=1$ and $\hat{u}_i(A)=1$) or $\tilde{u}_i(A) = *$ (i.e., $\underline{u}_i(A)=0$ and $\hat{u}_i(A)=1$). Therefore, the above labeled n-cube has the following labels:

$$L(A) = L^{f,i-1}_{mn}(A), \text{ if } u_i(A)=0$$

$$L(A) = L^{f,i-1}_{mx}(A), \text{ if } u_i(A)=1.$$

Next we shall prove that $C_n(u_1,u_2,\ldots,u_i, v_{i+1},\ldots,v_{R_f-1}, f)$ is an $NFS(R_f,f)$. This will, in turn, prove that any negative completion of $\tilde{u}_i$ with respect to $x_1,\ldots,x_n, u_1,\ldots,u_{i-1}$ produces a feasible $NFS^i(R_f,f)$. Suppose there is an inverse edge $\vec{AB}$ in $C_n(u_1,u_2,\ldots,u_i, v_{i+1},\ldots,v_{R_f-1}, f)$. Since $(u_1,\ldots,u_{i-1})$ is an $NFS(i-1,u_{i-1})$ which has no inverse edge, $u_j(A)=u_j(B)$ must hold for $j=1,\ldots,i-1$. Therefore, the following three cases must be considered:

(a) $u_i(A) = u_i(B) = 0$, but $L(A) > L(B)$.

This can never occur since from the above conclusion we have $L(A) = L_{mn}^{f,i-1}(A)$ and $L(B) = L_{mn}^{f,i-1}(B)$ which are produced by algorithm CMNL that guarantees the produced labeled n-cube has no inverse edge.

(b) $u_i(A) = u_i(B) = 1$, but $L(A) > L(B)$.

This can not occur since we have $L(A) = L_{mx}^{f,i-1}(A)$ and $L(B) = L_{mx}^{f,i-1}(B)$ which are produced by algorithm CMXL that guarantees the produced labeled n-cube has no inverse edge.

(c) $u_i(A) = 1$, $u_i(B) = 0$.

Since u_i is a negative completion of $\tilde{u}_i$ with respect to $x_1,\ldots,x_n, u_1,\ldots,u_{i-1}$, there can be no inverse edges in $C_{n+i-1}(u_i)$. However, $(a_1,\ldots,a_n, u_1(A),\ldots,u_{i-1}(A))$ and $(b_1,\ldots,b_n, u_1(B),\ldots, u_{i-1}(B))$ form a directed edge in $C_{n+i-1}(u_i)$ with $u_i(A)=1$ and $u_i(B)=0$. This contradicts the fact that $C_{n+1-1}(u_i)$ has no inverse edge. Therefore, $u_i(A)=1$ and $u_i(B)=0$ do not occur.

Summarizing (a), (b), and (c), it is clear that any negative completion u_i of $\tilde{u}_i$ will produce a feasible partially specified negative function sequence, $NFS^i(R_f,f) = (u_1,\ldots,u_i, u^*_{i+1},\ldots,u^*_{R_f-1}, f)$.

Consequently the theorem statement is proved.

Q.E.D.

6.4 Design Algorithm of Irredundant MOS Networks

Based on the algorithm for obtaining maximum permissible functions, we present our algorithm for the design of irredundant MOS networks as follows.

Algorithm 6.4.1 - Design of irredundant MOS networks with a minimum number of MOS cells for a given function f (DIMN).

Step 1 - Let $NFS^o(R_f,f) = (u^*_1,\ldots,u^*_{R_f-1}, f)$ and set i=1. (If R_f is not known at this step, it will be obtained after $\underline{NFS}^o(R_f,f)$ is obtained by applying CMNL in Step 2).

Step 2 - Use algorithm CMNL to obtain $\underline{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1}, \underline{u}_i,\ldots,\underline{u}_{R_f-1}, f)$.

Step 3 - Use algorithm CMXL to obtain $\widehat{NFS}^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1}, \hat{u}_i,\ldots,\hat{u}_{R_f-1}, f)$.

Step 4 - Obtain function $\tilde{u}_i$ by setting:

$\tilde{u}_i(A)=0$, if $\underline{u}_i(A) = \hat{u}_i(A)=0$;

$\tilde{u}_i(A)=1$, if $\underline{u}_i(A) = \hat{u}_i(A)=1$; and

$\tilde{u}_i(A)=*$, if $\underline{u}_i(A)=0$ and $\hat{u}_i(A)=1$.

Step 5 - Obtain an irredundant MOS cell configuration for $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ by an appropriate algorithm (e.g., Algorithms 5.1 and 5.2). Let u_i denote the function realized by this cell (u_i is now a completion of $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$).

Step 6 - If $i = R_f-1$, design an irredundant MOS cell configuration for f with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{R_f-1}$ and terminate this algorithm; otherwise set $i = i+1$ and go to Step 2.

The validity of this algorithm is proved by the following theorem.

Theorem 6.2 - An MOS network designed by algorithm DIMN (Algorithm 6.4.1) for function f is irredundant.

Proof - Let the network designed by algorithm DIMN for function f realize the following NFS:

$$NFS(R_f,f) = (u_1,\ldots,u_{R_f-1}, f).$$

Suppose that there are redundant FETs in the above network. Let g_i denote a gate (MOS cell) such that g_i has at least one redundant FET but every g_j, for $j=1,\ldots,i-1$, has no redundant FETs. Let $\overset{o}{NFS}(R_f,f)$ be the NFS realized by the MOS network after multiple extractions of these redundant FETs. Then

$$\overset{o}{NFS}(R_f,f) = (u_1,\ldots,u_{i-1}, \overset{o}{u}_i,\ldots,\overset{o}{u}_{R_f-1},f),$$

where $u_1,\ldots,u_{i-1}$ are unchanged because no FETs have been extracted from gates $g_1,\ldots,g_{i-1}$ of the network. Let $\tilde{u}_i$ be the function obtained by Step 4 of Algorithm 6.4.1 (DIMN) in the i-th iteration.

Clearly, $\overset{o}{u}_i$ is not a completion of $\tilde{u}_i$ because u_i was obtained by Step 5 which designs an irredundant MOS cell configuration for $\tilde{u}_i$ (that means that g_i will not realize a completion of $\tilde{u}_i$ if any FET is extracted from g_i). In other words $\overset{o}{u}_i$ is not a completion of $\tilde{u}_i$. Since $\tilde{u}_i$ is the maximum permissible function based on $NFS^{i-1}(u_1,\ldots, u_{i-1}, u^*_i,\ldots,u^*_{R_f-1}, f)$ (Theorem 6.3.1), $(u_1,\ldots,u_{i-1},\overset{o}{u}_i, u^*_{i+1},\ldots, u^*_{R_f-1}, f)$ can not be a feasible partially specified NFS for f (Definition 6.3.2) contradicting the assumption that $\overset{o}{NFS}(R_f,f) = (u_1,\ldots,u_{i-1}, \overset{o}{u}_i,\ldots,\overset{o}{u}_{R_f-1}, f)$ is an $NFS(R_f,f)$. Consequently, the MOS network designed by algorithm DIMN must be irredundant.

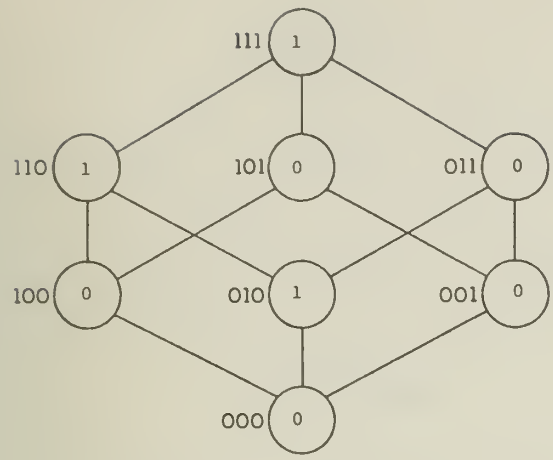
Q.E.D.

For better understanding, steps of algorithm MPF were explicitly written as Steps 2, 3, and 4 of algorithm DIMN. Steps 1, 2, 3, 4, and 6 of algorithm DIMN involve only simple labeling and comparisons. Step 5 requires an irredundant MOS cell configuration for an incompletely specified function. This can be obtained by using Algorithm 5.1 or 5.2. As explained in Section 5, these two algorithms are relatively simple compared to algorithms for obtaining irredundant disjunctive or conjunctive forms for an arbitrary function because Algorithms 5.1 and 5.2 are specially tailored for negative functions.

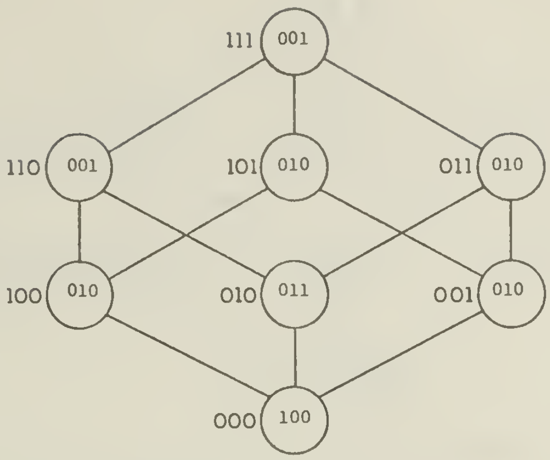
6.5 Examples

Two examples showing step-by-step applications of Algorithm 6.4.1 will be presented in order to help better understanding of the algorithm.

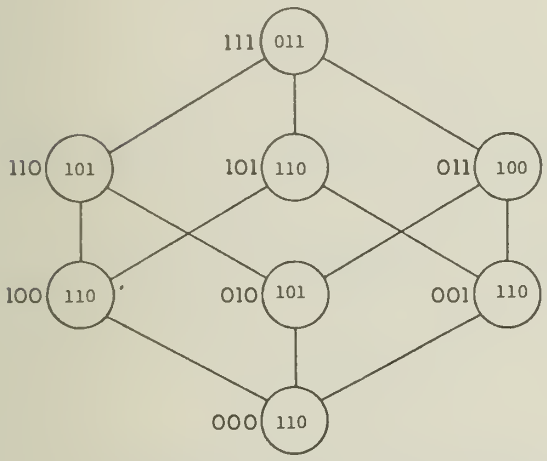
<u>Example 6.5.1</u> - Consider the function f considered in Example 3.1. The 3-cube with respect to f, i.e., $C_3(f)$, is shown in Fig. 6.5.1(a). Step 2 of Algorithm 6.4.1 obtains $NFS^o(3,f)$ as shown in (b) which is identical to the labeled 3-cube obtained by MNL shown in Fig. 3.2(b). Similarly, Step 3 obtains $\widehat{NFS}^o(3,f)$ as shown in (c). Step 4 then compares $\underline{u}_1$ and $\hat{u}_1$ and obtains $\tilde{u}_1$ and $\widetilde{NFS}^o(3,f)$ as shown in (d). The $NFS^1(3,f)_1 = (\bar{x}_1, u_2^*, f)$ in (e) is obtained by Step 5. Step 5 also obtains other two irredundant MOS cell configurations for u_1 and their corresponding $NFS^1(3,f)_2$ and $NFS^1(3,f)_3$ which will be discussed later. After Step 6, the algorithm returns to Step 2 and obtains $\underline{NFS}^1(3,f)_1$, $\widehat{NFS}^1(3,f)_1$, $\widetilde{NFS}^1(3,f)_1$, and $NFS^2(3,f)_1$ as shown in (f), (g), (h), and (i), respectively. Since $R_f = 3$, $NFS^2(3,f)_1$ is a completely specified negative function sequence with respect to x_1, x_2, and x_3. To finish the design, Step 6 of the algorithm obtains an irredundant MOS cell configuration for f. The completed design is shown in (s). As proved before, this MOS network has no redundant FETs. Furthermore, this network happens to consist of a minimum number of FETs since 3 load FETs and 5 driver FETs are the minimum number of FETs required by function f which can be proved as follows. Based on algorithm MNL, we know that function f requires three MOS cells which means that there are three load FETs and at least two interconnections between cells in any minimum MOS network for f. Since f is a function of exactly three variables, there are at least three connections from external variables (x_1,x_2, and x_3) to the three MOS cells. Therefore, in a minimum



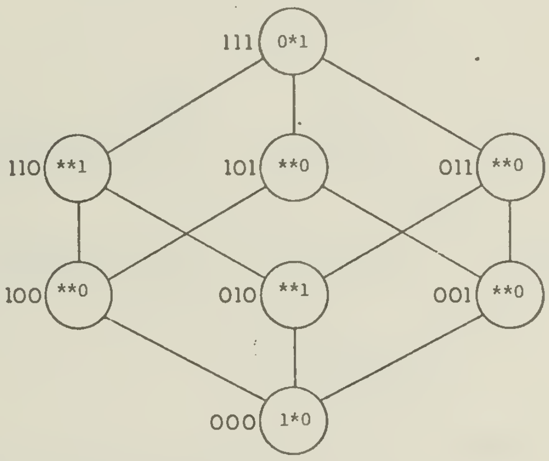
(a) $C_3(f)$ for $f = x_1x_2 \vee x_2\bar{x}_3$.



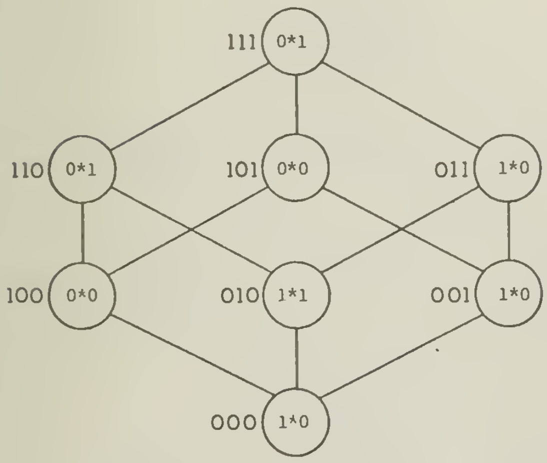
(b) $\underline{NFS}^o(3,f) = (\underline{u}_1, \underline{u}_2, f)$.



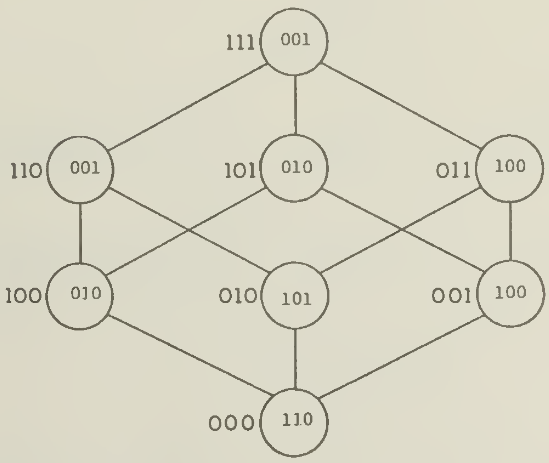
(c) $\widehat{NFS}^o(3,f) = (\hat{u}_1, \hat{u}_2, f)$.



(d) $\widetilde{NFS}^o(3,f) = (\tilde{u}_1, u_2^*, f)$.

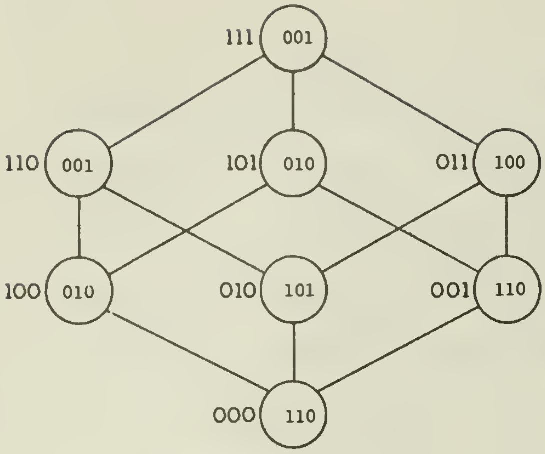


(d) $NFS^1(3,f)_1 = (\bar{x}_1, u_2^*, f)$.

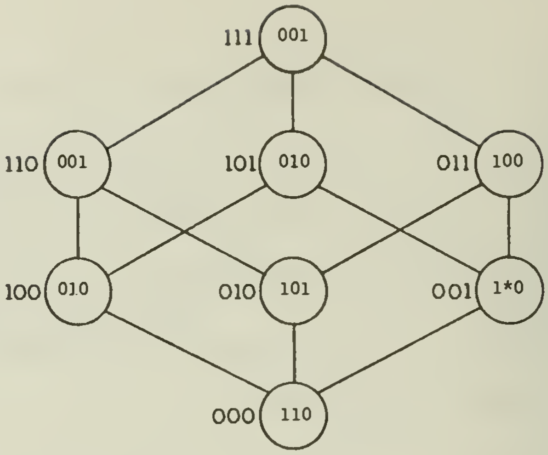


(f) $\underline{NFS}^1(3,f)_1 = (\bar{x}_1, \underline{u}_2, f)$.

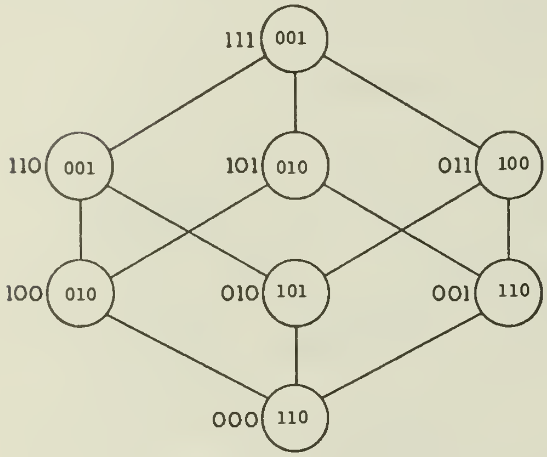
Fig. 6.5.1 Example 6.5.1.



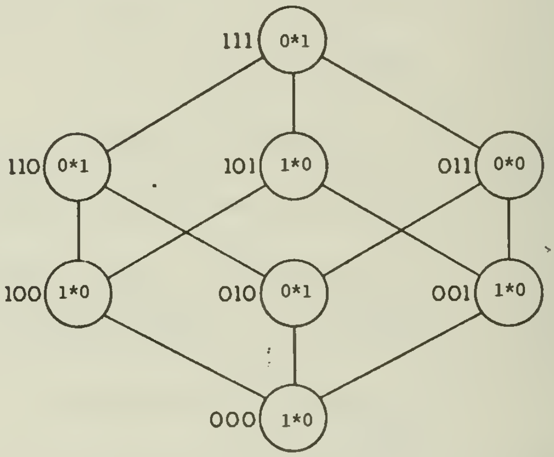
(g) $\hat{NFS}^1(3,f)_1 = (\bar{x}_1, \hat{u}_2, f)$.



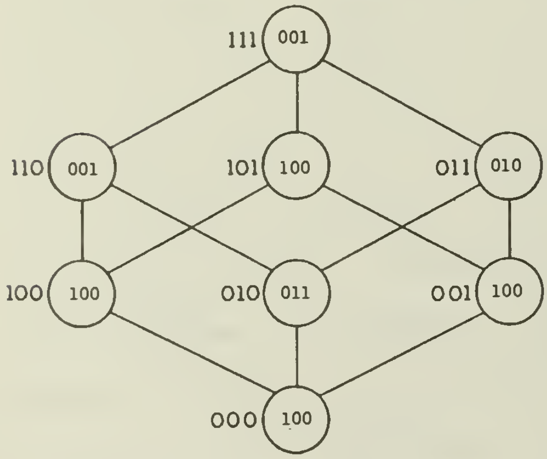
(h) $\tilde{NFS}^1(3,f)_1 = (\bar{x}_1, \tilde{u}_2, f)$.



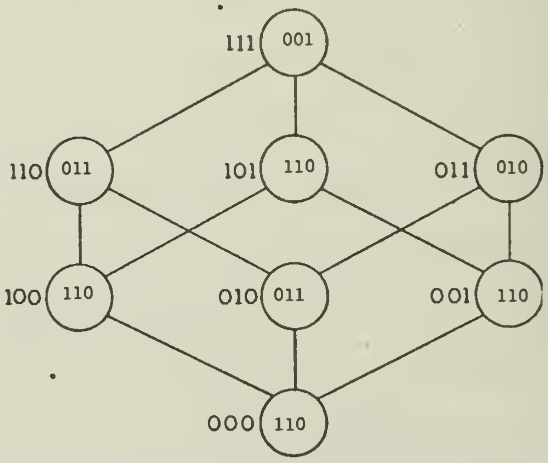
(i) $NFS^2(3,f)_1 = NFS(3,f)_1 = (\bar{x}_1,\bar{x}_2,f)$



(j) $NFS^1(3,f)_2 = (\bar{x}_2,u_2^*,f)$ obtained from (d).

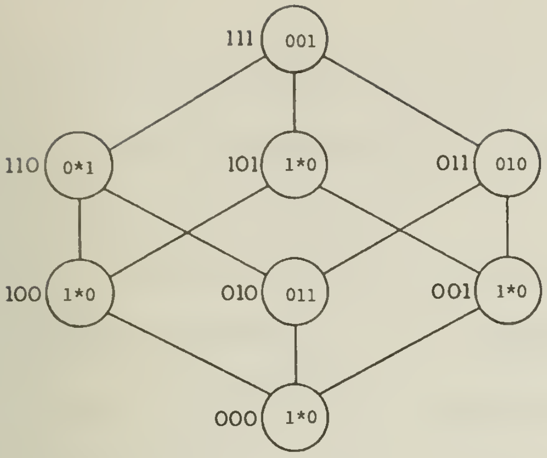


(k) $\underline{NFS}^1(3,f)_2 = (\bar{x}_2, \underline{u}_2, f)$.

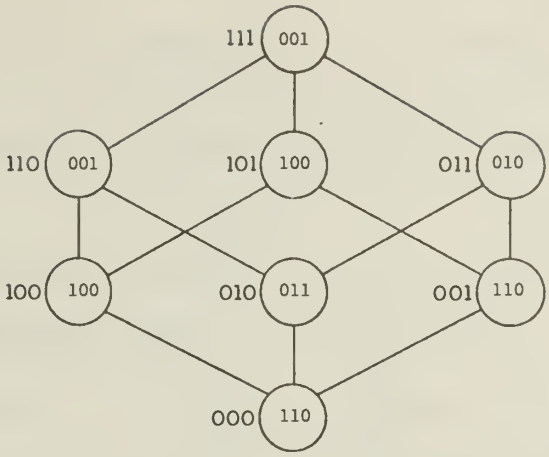


(ℓ) $\widehat{NFS}^1(3,f)_2 = (\bar{x}_2, \hat{u}_2, f)$.

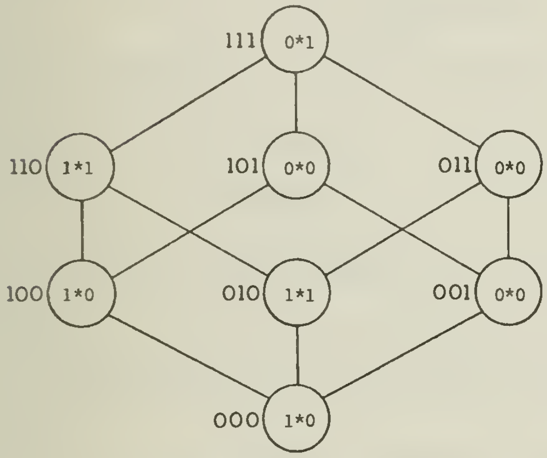
Fig. 6.5.1 (Continued)



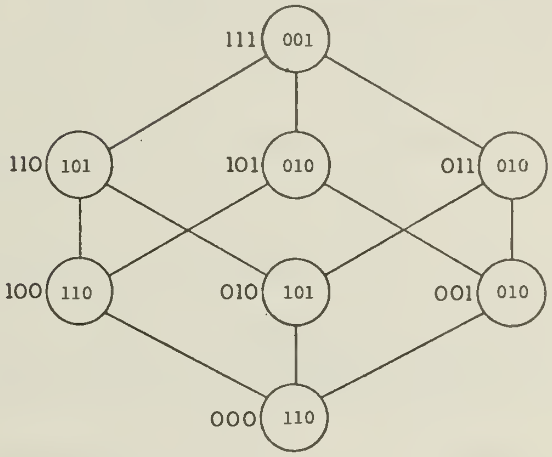
(m) $\widetilde{\mathrm{NFS}}^1(3,f)_2 = (\bar{x}_2, \tilde{u}_2, f)$.



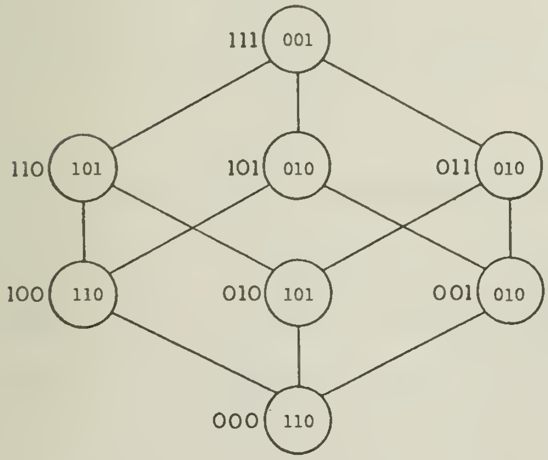
(n) $\mathrm{NFS}^2(3,f)_2 = \mathrm{NFS}(3,f)_2 = (\bar{x}_2, \bar{x}_1, f)$.



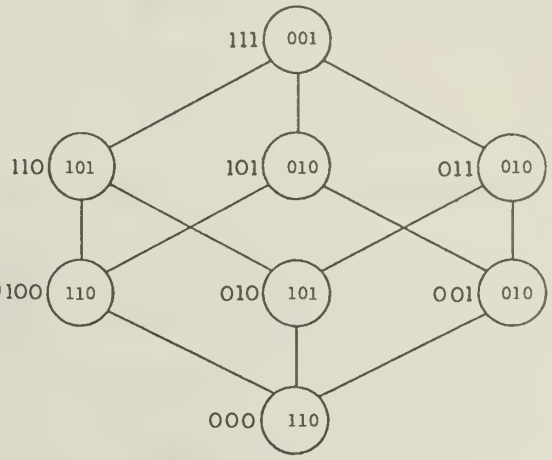
(o) $\mathrm{NFS}^1(3,f)_3 = (\bar{x}_3, u_2^*, f)$ obtained from (d).



(p) $\underline{\mathrm{NFS}}^1(3,f)_3 = (\bar{x}_3, \underline{u}_2, f)$.

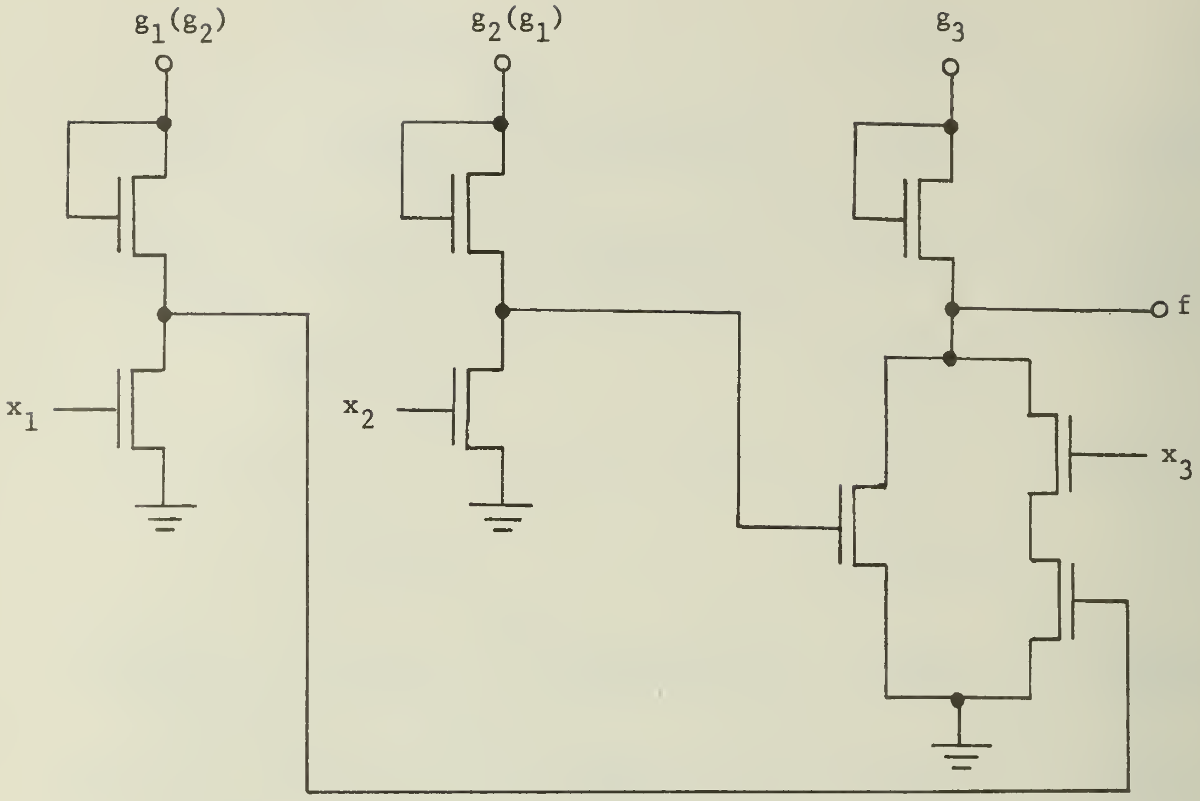


(q) $\widehat{\mathrm{NFS}}^1(3,f)_3 = (\bar{x}_3, \hat{u}_2, f)$.

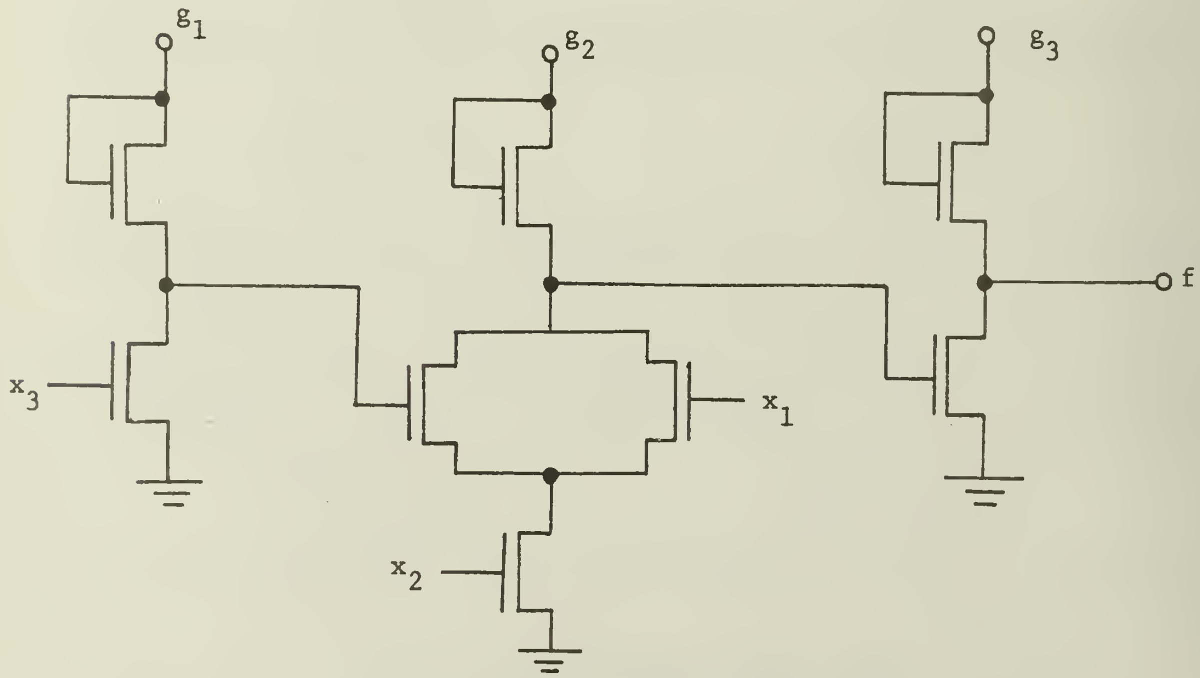


(r) $\widetilde{\mathrm{NFS}}^1(3,f)_3 = \mathrm{NFS}(3,f) = (\bar{x}_3, \bar{f}, f)$.

<u>Fig. 6.5.1</u> (Continued)



(s) Irredundant MOS network corresponding to NFS(3,f)'s in (i) and (n).



(t) Irredundant MOS network corresponding to NFS$(3,f)_3$ in (r).

Fig. 6.5.1 (Continued)

MOS network for f, there are at least three MOS cells and five connections which means that f requires at least three load FETs and five driver FETs.

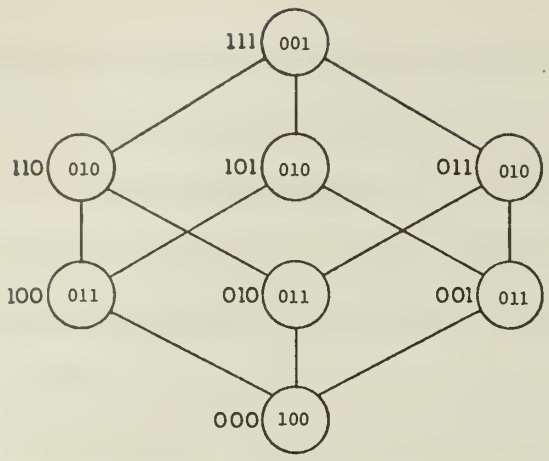
As mentioned above, Step 5 of the algorithm can also obtain $NFS^1(3,f)_2 = (\bar{x}_2, u_2^*, f)$ or $NFS^1(3,f)_3 = (\bar{x}_3, u_2^*, f)$ as shown in Figs. 6.5.1(j) and (o), respectively. From (j), the algorithm obtains $\underline{NFS}^1(3,f)_2$, $\widehat{NFS}^1(3,f)_2$, $\widetilde{NFS}^1(3,f)_2$, and $NFS^2(3,f)_2$ as shown in (k), (l), (m), and (n), respectively. $NFS(3,f)_2$ and $NFS(3,f)_1$ are indistinguishable, since the permutation of u_1 and u_2 in $NFS(3,f)_2$ results in $NFS(3,f)_1$. The corresponding MOS network for both NFS(3,f)'s is shown in Fig. 6.5.1(s). Similarly, from (o), the algorithm obtains $\underline{NFS}^1(3,f)_3$, $\widehat{NFS}^1(3,f)_3$, and $\widetilde{NFS}^1(3,f)_3$, as shown in (p), (q), and (r), respectively. Since $\tilde{u}_2$ in $\widetilde{NFS}^1(3,f)_3$ has no unspecified value with respect to x_1, x_2, and x_3, $\widetilde{NFS}^1(3,f)_3 = NFS(3,f)_3$. The MOS network corresponding to $NFS(3,f)_3$ is shown in (t), which also consists of a minimum number of FETs for function f as we can prove in the same manner as before.

As evident from Example 6.5.1, Steps 2, 3, and 4 are deterministic. By this we mean that given a $NFS^{i-1}(R_f,f)$ subsequent $\underline{NFS}^{i-1}(R_f,f)$, $\widehat{NFS}^{i-1}(R_f,f)$, and $\widetilde{NFS}^{i-1}(R_f,f)$ are uniquely determined. On the other hand, Step 5 is generally non-deterministic because we may obtain more than one irredundant MOS cell configurations for a given $\tilde{u}_i$. In our simple example, when Step 5 of the algorithm is reached for the first time, three irredundant MOS cell configurations for $\tilde{u}_1$ can be found. If we are only interested in one irredundant MOS network design for f, any one of the three can be used. No matter which particular irredundant MOS cell configuration is chosen, the designed MOS network will be irredundant.

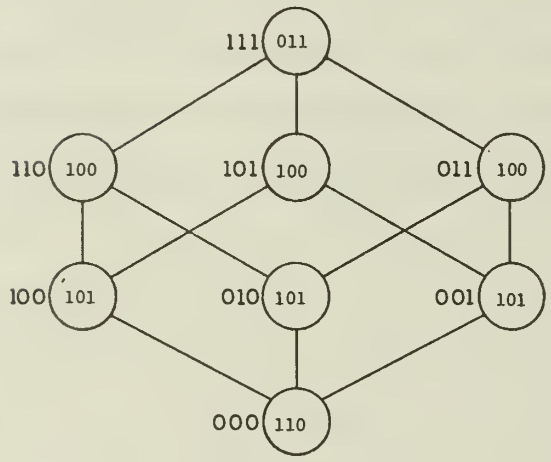
It should be noted that this algorithm usually requires R_f-1 iterations in order to obtain $NFS^{i-1}(R_f,f)$, $\underline{NFS}^{i-1}(R_f,f)$, $\widehat{NFS}^{i-1}(R_f,f)$, $\widetilde{NFS}^{i-1}(R_f,f)$ and $NFS^{i}(R_f,f)$ in each iteration for $i=1,\ldots,R_f-1$. However, if $\widetilde{NFS}^{i-1}(R_f,f)$ becomes a completely specified function with respect to $x_1,\ldots,x_n$, for some $i < R_f$ (i.e., $\underline{NFS}^{i-1}(R_f,f) = \widehat{NFS}^{i-1}(R_f,f)$ holds), then $\widetilde{NFS}^{i-1}(R_f,f) = NFS^{i}(R_f,f) = NFS(R_f,f)$ must hold. In this case, the algorithm only requires i iterations. Even in this case, however, Step 5 of the algorithm must be executed R_f-i-1 additional times in order to obtain the irredundant MOS cell design for $u_{i+1},\ldots,u_{R_f-1}$ (the cell design for f is taken care of by Step 6). In Example 6.5.1, when we choose $u_1 = \bar{x}_3$ as the irredundant MOS cell configuration for $\tilde{u}_1$, $\underline{NFS}^{1}(3,f)_3$ and $\widehat{NFS}^{1}(3,f)_3$ become the same. Therefore in this case, $\underline{NFS}^{1}(3,f)_3 = \widehat{NFS}^{1}(3,f)_3 = NFS(3,f)_3$.

Another point to be noted is that the exhaustion of all alternatives in Step 5 may not give all irredundant MOS network designs for the given function by the following reason. First, the algorithm does not give a solution which is irredundant but consists of more cells than the minimum number required. Next, some irredundant MOS network with a minimal number of MOS cells may not be produced by this algorithm. The following example demonstrates this fact.

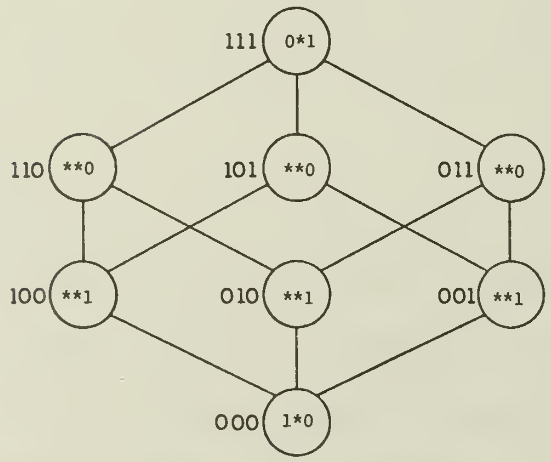
Example 6.5.2 - Consider the odd-parity function of three variables, $f = x_1 \oplus x_2 \oplus x_3$. According to Algorithm 6.4.1, $\underline{NFS}^o(3,f)$, $\widehat{NFS}^o(3,f)$, and $\widetilde{NFS}^o(3,f)$ are obtained as shown in Fig. 6.5.2(a), (b), and (c), respectively. From $\tilde{u}_1$ in $\widetilde{NFS}^o(3,f)$, $u_1 = \bar{x}_1$, $\bar{x}_2$, or $\bar{x}_3$ are obtained as the only irredundant MOS cell configurations for $\tilde{u}_1$. This means that every solution obtained by Algorithm 6.4.1 will contain an MOS cell which works as an inverter for one of the input variables. However, the MOS network shown in Fig. 6.5.2(d) realizes f and is irredundant but contains no such MOS cell. Therefore, some irredundant MOS networks with a minimum number of cells may not be produced by Algorithm 6.4.1.



(a) $\underline{NFS}^\circ(3,f) = (\underline{u}_1,\underline{u}_2,f)$ for $f = x_1 \oplus x_2 \oplus x_3$.

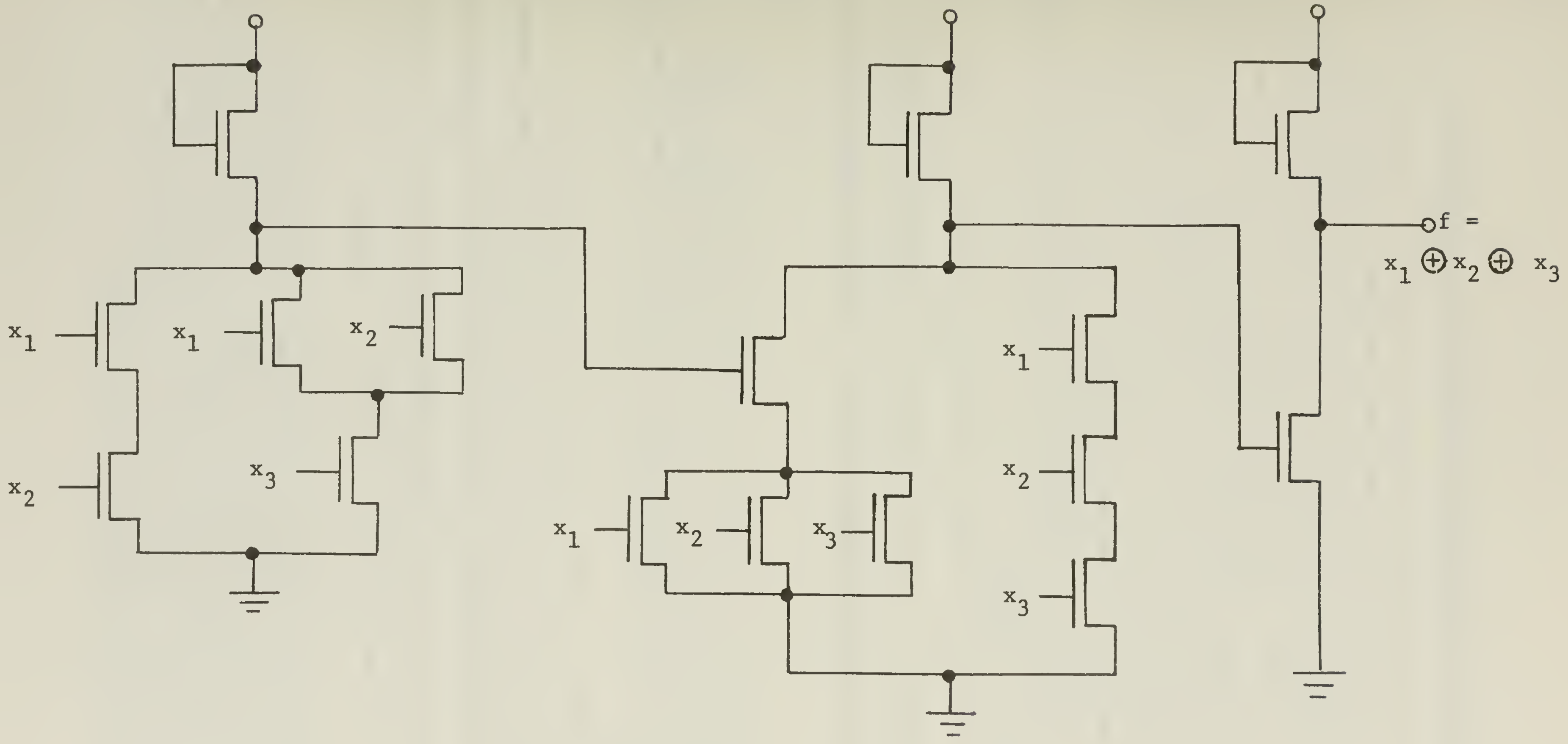


(b) $\widehat{NFS}^\circ(3,f) = (\hat{u}_1,\hat{u}_2,f)$.



(c) $\widetilde{NFS}^\circ(3,f) = (\tilde{u}_1,u_2^*,f)$.

Fig. 6.5.2 Example 6.5.2.



(d) An irredundant MOS network for f which can not be produced by Algorithm 6.4.1.

Fig. 6.5.2 (Continued)

7. DIAGNOSTIC PROPERTIES OF IRREDUNDANT MOS NETWORKS

In the previous section, we have presented an algorithm to design irredundant MOS networks for a given completely specified function. The term "irredundant" means that no FET in the network can be extracted from the network without changing the output of the network as defined in Section 5. This property indicates that the designed network is a "diagnosable network." This section will discuss the diagnostic properties of the networks designed according to Algorithm 6.4.1 (DIMN).

Definition 7.1 - An FET is said to be in the stuck-at-short failure mode (or stuck-at-short fault) if this FET becomes permanently conductive.

Definition 7.2 - An FET is said to be in the stuck-at-open failure mode (or stuck-at-open fault) if this FET becomes permanently nonconductive.

The stuck-at-short and stuck-at-open faults can be represented by the FET input permanently stuck-at-1 and stuck-at-0, respectively. This fault model has its practical justifications since it is supported by an engineering analysis [SK 69] as discussed in [Pai 73].

Definition 7.3 - An MOS network is said to be diagnosable if every possible single or multiple stuck-at-short and/or stuck-at-open faults in the network can be detected by comparing the output of the faulty network with the faultless network output.

This definition is more convenient in practice than the one used in [Pai 73] where a network is said to be diagnosable if every single and multiple stuck-at-short and/or stuck-at-open faults can be detected by comparing the output of each faulty cell with the faultless one. If we want to diagnose a diagnosable network defined by [Pai 73], test terminals must be provided at every cell in the network in order to detect all possible stuck-at-short and/or stuck-at-open faults in all the cells in the network. On the other hand, if a network is diagnosable in the sense of Definition 7.3, all possible faults can be detected at the network output without the use of extra terminals for testing.

<u>Theorem 7.1</u> - An MOS network is diagnosable if and only if it is irredundant.

<u>Proof</u> - From Definitions 5.1, 7.1 and 7.2, it is obvious that the stuck-at-short fault of an FET in an MOS network is equivalent to the s-extraction of the FET from the network. Similarly, the stuck-at-open fault of an FET in an MOS network is equivalent to the o-extraction of the FET from the network. From Definitions 5.2, 5.4, and 7.3, therefore, it is obvious that an MOS network is diagnosable if and only if it is irredundant.

Q.E.D.

Let symbol N denote an MOS network of n variables which consists of k FETs $D_1, D_2, \ldots, D_k$. A faulty network of N will be denoted by N(F) where F is a set of faulty FETs with their respective failure modes expressed as D_i^s meaning D_i stuck-at-short or D_i^o meaning D_i

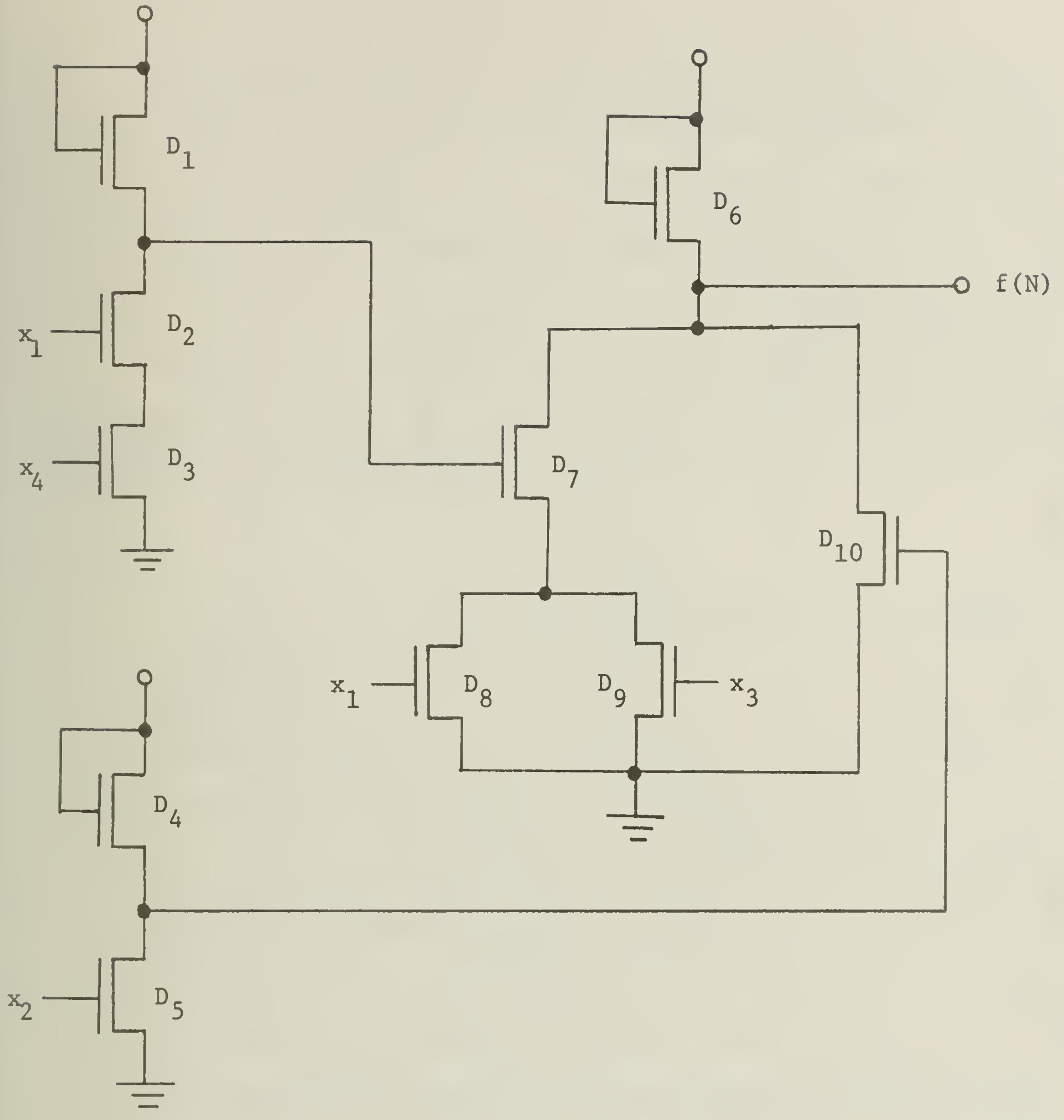
stuck-at-open. Also let $f(N,A)$ and $f(N(F),A)$ for input $A \varepsilon V_n$ be the function value produced by networks N and N(F), respectively, for input $A \varepsilon V_n$. $f(N)$ and $f(N(F))$ may be used when we are not concerned with any specific input vector.

<u>Definition 7.4</u> - A <u>test</u>, A, for a network N(F) with a single or multiple fault F is an input vector A in V_n such that $f(N,A) \neq f(N(F),A)$.

<u>Example 7.1</u> - Fig. 7.1(a) shows a network N for a four-variable function $f(N) = x_1x_2x_4 \vee \bar{x}_1x_2\bar{x}_3$. Let $F = (D_3^s , D_8^o)$ be a multiple fault of N. Fig. 7.1(b) shows network N(F) whose output realizes $f(N(F)) = x_1x_2 \vee x_2\bar{x}_3$. Comparing these two functions, it is clear that the input vectors (1100) and (1110) are the only tests for N(F) since $f(N, (1100)) = 0$ and $f(N(F), (1100)) = 1$, $f(N, (1110)) = 0$ and $f(N(F), (1110)) = 1$, and for all other input vectors B we have $f(N,B) = f(N(F),B)$.

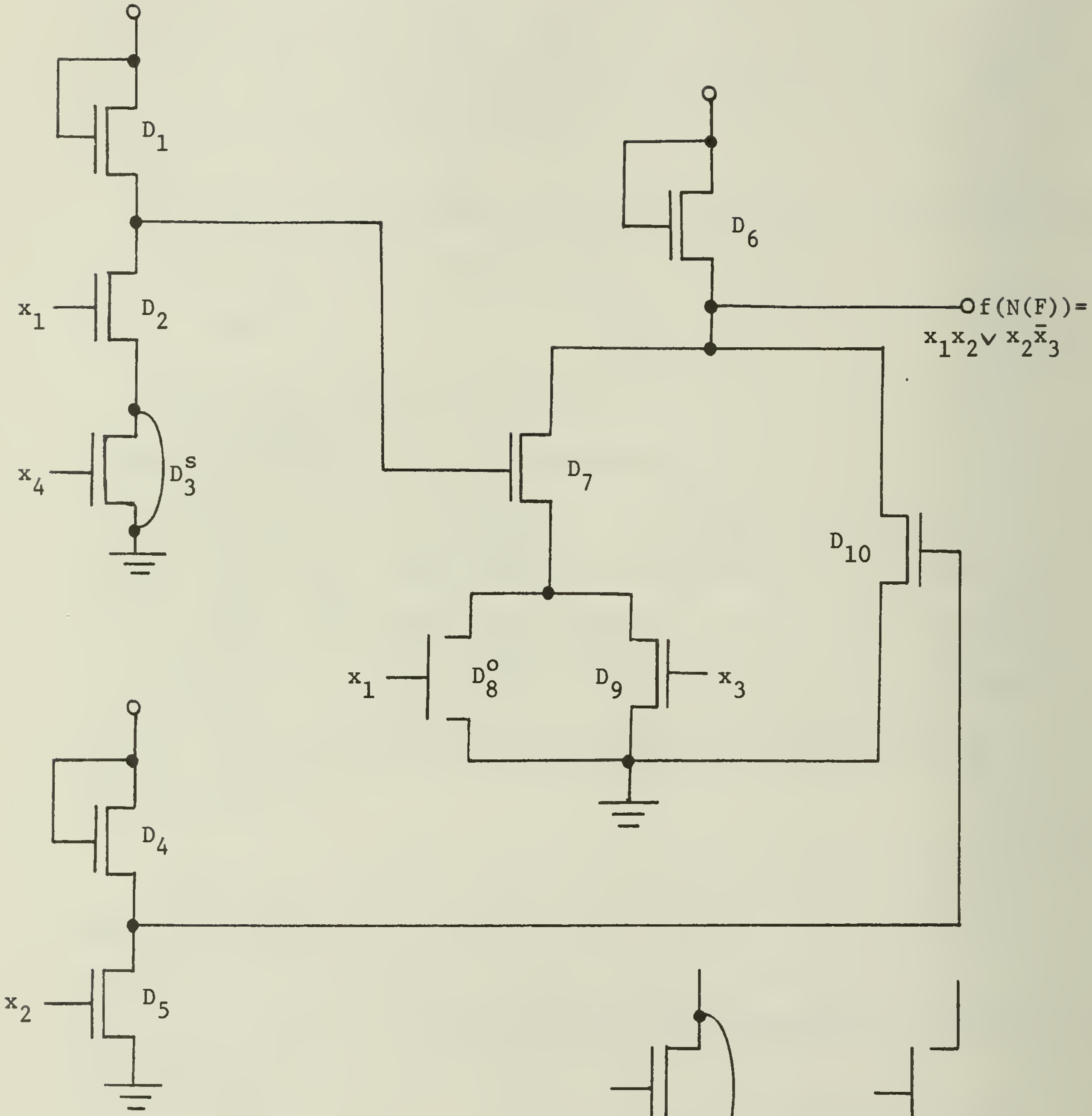
Definition 7.4 defines tests for a particular fault (either single or multiple) in a network. If we consider a set of faults $\Phi = \{F_1,\ldots,F_k\}$ (the network can have only one single or multiple fault F_i at any given moment), we can define a sufficient test set as follows.

<u>Definition 7.5</u> - A <u>sufficient test set</u> Δ for a class of faults Φ in an MOS network N is a set of input vectors $\Delta \subseteq V_n$ such that for each $F \varepsilon \Phi$ there exists an input vector $A \varepsilon \Delta$ satisfying $f(N,A) \neq f(N(F),A)$, in other words, A is a test for N(F).

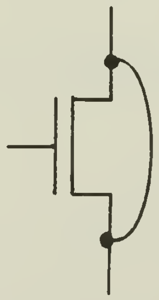


(a) Network N for function $f(N) = x_1x_2x_4 \vee \bar{x}_1x_2\bar{x}_3$.

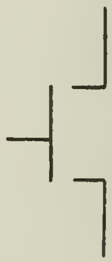
Fig. 7.1 Example 7.1.



(b) Network N ($\{D_3^s, D_8^o\}$).



FET stuck-at-short



FET stuck-at-open

Fig. 7.1 (Continued)

The algorithm DIMN presented in Section 6 designs irredundant MOS networks for a given function. According to Theorem 7.1, these networks are diagnosable, i.e., for every possible single or multiple fault F in such a network N, there exists a test A such that $f(N,A) \neq f(N(F),A)$. This fact is stated in the following corollary.

Corollary 7.1 - The set of all 2^n input vectors, V_n, is a sufficient test set for all possible single or multiple stuck-at-short and/or stuck-at-open faults in a network designed by Algorithm 6.4.1 (DIMN).

Although V_n is a trivial sufficient test set for irredundant networks, it may not be a sufficient test set for other networks. For example, reference [Pai 73] presented a procedure to design two-level diagnosable[†] MOS networks which requires auxiliary test points in order to make all single and multiple faults in the network fully detectable. If a network requires auxiliary test points, the total number of tests may exceed the total number 2^n of input vectors (V_n) since the output cell generally has more than n inputs. In an example given in [Pai 73] the output cell of the network of three variable which is designed by the two-level diagnosable MOS network design procedure requires nine tests even though the total number of input vectors for that network is only eight.

[†]The meaning of the term "diagnosable" used in [Pai 73] is different from the one defined in Definition 7.3 of this paper, as noted earlier in this section.

Next, we will present some diagnostic properties possessed by the networks designed by Algorithm 6.4.1 (DIMN).

Definition 7.6 - Set $S_E(f)$ is the set of inverse edges in $C_n(f)$, i.e., $\vec{AB} \varepsilon S_E(f)$ if and only if $\vec{AB} \varepsilon C_n$, and $f(A) = 1$ and $f(B) = 0$.

Definition 7.7 - The characteristic input set $S_C(f)$ for a function f is a subset of input vectors such that $A \varepsilon S_C(f)$ if and only if $\vec{AB} \varepsilon S_E(f)$ or $\vec{BA} \varepsilon S_E(f)$ for some $B \varepsilon V_n$.

Example 7.2 - Consider the function $f = x_1x_2 \vee x_2\bar{x}_3$ in Example 6.5.1. Fig. 7.2 shows the labeled 3-cube for f where inverse edges are shown in bold lines.

$$S_E(f) = \{\overrightarrow{(111)(101)}, \overrightarrow{(111)(011)}, \overrightarrow{(110)(100)}, \overrightarrow{(010)(000)}\}$$

The characteristic input set of function f is

$$S_C(f) = \{(111), (101), (011), (110), (100), (010), (000)\}.$$

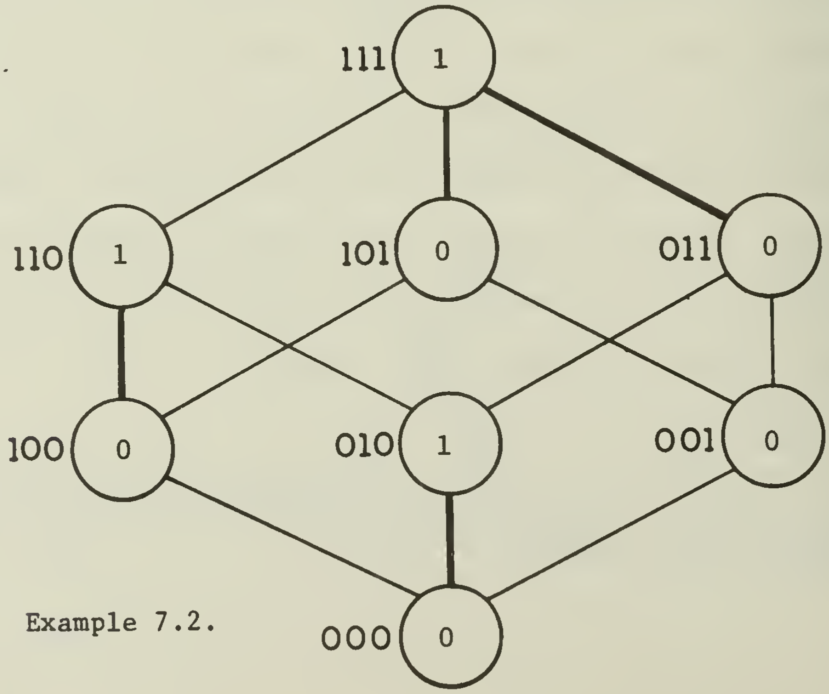


Fig. 7.2 Example 7.2.

The following lemma is based on the definition of $S_E(f)$. This lemma leads to an important property of irredundant networks stated in Theorem 7.2.

Lemma 7.1 - Given two functions of n variables f_1 and f_2 such that $S_E(f_1) \subseteq S_E(f_2)$, if $(u_1,\ldots,u_{R-1}, f_2)$ is a negative function sequence NFS(R,f_2) for f_2, then the sequence of functions $(u_1,\ldots, u_{R-1}, f_1)$ must also be a negative function sequence NFS(R,f_1) for f_1, where $u_1,\ldots,u_{R-1}$ are the same functions as in NFS(R,f_2).

Proof - Let us consider the labeled n-cube with respect to $(u_1,\ldots,u_{R-1}, f_1)$, $C_n(u_1,\ldots,u_{R-1}, f_1)$. If $(u_1,u_2,\ldots,u_{R-1}, f_1)$ is not an NFS(R,f_1), there must exist an inverse edge $\vec{AB}$ in $C_n(u_1,\ldots,u_{R-1}, f_1)$ such that

$$\ell(A; u_1,\ldots,u_{R-1}, f_1) > \ell(B; u_1,\ldots,u_{R-1}, f_1). \qquad (7.1)$$

However, since $(u_1,\ldots,u_{R-1}, f_2)$ is an NFS(R,f_2),

$$\ell(A; u_1,\ldots,u_{R-1}, f_2) \leq \ell(B; u_1,\ldots,u_{R-1}, f_2) \qquad (7.2)$$

must hold. From these two inequalities, we have $u_i(A) = u_i(B)$ for $i=1,\ldots,R-1$ and $f_1(A) = 1$, $f_1(B) = 0$ for the following reason. Assume $u_1(A)=u_1(B),\ldots,u_{i-1}(A)=u_{i-1}(B)$ but $u_i(A) \neq u_i(B)$. Because of (7.1), $u_i(A) > u_i(B)$ must hold, contradicting (7.2). Therefore $u_i(A) = u_i(B)$ for $i=1,\ldots,R-1$ must hold. Then from (7.1), $f_1(A) = 1$, $f_1(B) = 0$ must hold.

This means that $\vec{AB}$ is an inverse edge in $C_n(f_1)$, i.e., $\vec{AB} \in S_E(f_1)$. Because of the assumption that $S_E(f_1) \subseteq S_E(f_2)$, $\vec{AB}$ must be in $S_E(f_2)$, i.e., $f_2(A) = 1$ and $f_2(B) = 0$ must hold. This results in $\ell(A; u_1,\ldots, u_{R-1}, f_2) > \ell(B; u_1,\ldots,u_{R-1}, f_2)$ which contradicts (7.2). Consequently, $(u_1,\ldots,u_{R-1}, f_1)$ must be an NFS(R,f_1).

Q.E.D.

Lemma 7.1 asserts that for any function f_1 such that $S_E(f_1) \subseteq S_E(f_2)$, we can obtain an MOS network for f_1 by redesigning only the output cell of an MOS network for f_2.

<u>Definition 7.8</u> - A <u>POCF (Pure Output Cell Fault)</u>, F_p, for an MOS network N is a fault which involves FET(s) only in the output cell. In other words, for every D_i^s or $D_i^o \in F_p$, D_i is an FET in the output cell of network N.

<u>Definition 7.9</u> - An <u>NPOCF (Non-Pure Output Cell Fault)</u>, F_n, for an MOS network N is a fault which involves at least one FET which is not in the output cell, i.e., there exist at least one D_i^s or $D_i^o \in F_n$ such that D_i is not in the output cell.

Let Φ_p denote the set of all possible POCF for network N, and Φ_n the set of all possible NPOCF for network N. It is obvious that Φ_t, the set of all possible single and multiple faults for N is the union of Φ_p and Φ_n, i.e.,

$$\Phi_t = \Phi_p \cup \Phi_n .$$

Example 7.3 - Consider the network considered in Example 7.1. For the network N shown in Fig. 7.1(a), $F = (D_8^o)$ and $F = (D_7^s, D_8^o)$ are POCFs since both D_7 and D_8 are in the output cell. On the other hand, $F = (D_3^s)$, $F = (D_3^s, D_8^o)$, and $F = (D_3^s, D_5^o)$ are NPOCFs since D_3 is an FET not in the output cell.

Theorem 7.2 - The characteristic input set $S_C(f)$ for a function f is a sufficient test set for Φ_n, the set of all possible non-pure output cell faults for a network N designed by Algorithm 6.4.1 for f.

Proof - Suppose $S_C(f)$ is not a sufficient test set for Φ_n for N. This means that some fault $F \varepsilon \Phi_n$ for N can not be detected by any input vectors in $S_C(f)$. In other words, $f(N(F),A) = f(N,A)$ for every $A \varepsilon S_C(f)$, which means $S_E(f) \subseteq S_E(f(N(F)))$. Now let g_i denote the cell with a smallest subscript which has faulty FETs. Since F is a NPOCF, $i < R_f$ must hold. Then the negative function sequence $(u_1, \ldots, u_{R_f-1}, f)$ realized by N changes to the negative function sequence $(u_i, \ldots, u_{i-1}, \overset{o}{u}_i, \ldots, \overset{o}{u}_{R_f-1}, f(N(F)))$ realized by N(F). Now let us compare $NFS(R_f, f(N(F))) = (u_1, \ldots, u_{i-1}, \overset{o}{u}_i, \ldots, \overset{o}{u}_{R_f-1}, f(N(F)))$ and the sequence of functions $(u_1, \ldots, u_{i-1}, \overset{o}{u}_i, \ldots, \overset{o}{u}_{R_f-1}, f)$, where only the last function is changed. Since $S_E(f) \subseteq S_E(f(N(F)))$ holds, $(u_1, \ldots, u_{i-1}, \overset{o}{u}_i, \ldots, \overset{o}{u}_{R_f-1}, f)$ must be an NFS(R,f) according to Lemma 7.1. In other words, by properly re-designing only the output cell of the faulty network N(F), we can obtain an MOS network for f. However, since N is designed by Algorithm 6.4.1, and, more specifically, since cell g_i is irredundant with respect to $\tilde{u}_i$, $\overset{o}{u}_i$ is not a completion of $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$, where $\tilde{u}_i$ is the maximum permissible function for u_i^* in $NFS^{i-1}(R_f,f) = (u_1,\ldots,u_{i-1}, u_i^*,\ldots,u_{R_f-1}^*, f)$. Therefore, by Definition 6.3.2 and Theorem 6.3.1, $(u_1,\ldots,u_{i-1}, \overset{o}{u}_i,\ldots,\overset{o}{u}_{R_f-1}, f)$ can not be an NFS, contradicting an earlier result that $(u_1,\ldots,u_{i-1}, \overset{o}{u}_i,\ldots,\overset{o}{u}_{R_f-1}, f)$ is an $NFS(R_f,f)$. Consequently, $S_C(f)$ is a sufficient test set for Φ_n of N.

Q.E.D.

$S_C(f)$ is usually a redundant test set for faults in Φ_n for a network N designed by Algorithm 6.4.1 for function f. Section 8 will give an example to illustrate that after eliminating some vectors from $S_C(f)$ the remaining vectors in $S_C(f)$ is still a sufficient test set for all the faults in Φ_n for N.

The test set $S_C(f)$ for Φ_n is a "universal" test set in the following sense. As discussed in Section 6, Algorithm 6.4.1 may give more than one irredundant MOS network for a given function. The proof of Theorem 7.2 is not restricted to one particular network but can be applied to every network designed by Algorithm 6.4.1. In this sense, $S_C(f)$ is a <u>universal test set</u> for the set of all NPOCFs for all the networks designed by Algorithm 6.4.1. In other words, no matter which particular irredundant MOS cell configuration we choose in Step 5 in each iteration of Algorithm 6.4.1, $S_C(f)$ is a sufficient test set for all faults in Φ_n for the resultant network.

Faults in Φ_p may not be detected with tests in $S_C(f)$ only, but finding a sufficient test set for Φ_p is relatively easy since all faulty FETs are located in the output cell. In other words, when we want to derive a test set for faults in Φ_p only, all inputs to the output cell are correct and then the test set can be obtained based on the configuration of the output cell only. A test generation procedure given in [Pai 73] may be applied with minor modifications.

Reference [Ake 73] discusses the universal test sets[†] for logic networks consisting of AND/OR gates. Based on the fact that AND/OR networks can realize only functions which are positive with respect to the input literals, it was proved that there exists for each function f a universal test set which can detect all single and multiple stuck-at-0 and/or stuck-at-1 faults in any AND/OR networks realizing function f. In the case of MOS networks, since each cell can realize only functions which are negative with respect to the inputs of the cell, a similar discussion can be applied here. Therefore, for the set of pure output cell faults there exists a universal test set which is determined by the given function f realized by the output cell and the functions realized by other cells which constitute the inputs of the output cell. This universal test set will be independent of the output cell configuration.

[†]The term "universal test set" in [Ake 73] means that the test set can detect all single or multiple faults in any AND/OR networks for a given function regardless how these networks are designed. In this sense, it is different from what "universal test set" in the current thesis means. However, [Ake 73] did not discuss the irredundancy of the AND/OR networks.

Example 7.4 - Consider the function $f = x_1x_2 \vee x_2\bar{x}_3$ for which Algorithm 6.4.1 designed two irredundant MOS networks shown in Fig. 6.5.1(s) and (t) in Example 6.5.1. For both networks, $S_C(f) = \{(111), (101), (011), (110), (100), (010), (000)\}$ (see Example 7.2) can detect all non-pure output cell faults. This can be demonstrated exhaustively, but is omitted here.

8. EXTENSION TO MOS NETWORK FOR INCOMPLETELY SPECIFIED FUNCTION

In the previous sections, an algorithm for the design of irredundant MOS networks for a completely specified single-output function has been presented. In this section we will modify the algorithm to be applicable to incompletely specified functions.

The extension of the algorithms to the case of incompletely specified functions is straightforward. Let $\tilde{f}$ denote an incompletely specified function. For each input vector $A \varepsilon V_n$, $\tilde{f}(A)$ has the value of 0, 1, or * where * indicates a don't care, i.e., $\tilde{f}(A)$ is not specified. Let f denote a completion of $\tilde{f}$, i.e., f satisfies $f(A) = \tilde{f}(A)$ for every $A \varepsilon V_n$ such that $\tilde{f}(A) \neq *$, and $f(A)=0$ or 1 for each A such that $\tilde{f}(A)=*$.

It is easy to modify algorithm MNL for an incompletely specified function, as discussed in [NTK 72]. The only change necessary is to assign proper values to don't cares in $\tilde{f}$ such that each vertex is assigned a minimum possible label satisfying that the labeled n-cube has no inverse edges. This can be achieved by simply modifying algorithm MNL (Algorithm 3.1) as follows.

<u>Algorithm 8.1</u> - Algorithm MNL for an incompletely specified function $\tilde{f}$.

<u>Step 1</u> - If $\tilde{f}(I) = *$, assign $L_{mn}^{\tilde{f}}(I) = 0$; otherwise assign $L_{mn}^{\tilde{f}}(I) = \tilde{f}(I)$.

<u>Step 2</u> - When $L_{mn}^{\tilde{f}}(A)$ is assigned to each vertex A of weight $w(1 \leq w \leq n)$ of C_n, assign as $L_{mn}^{\tilde{f}}(B)$ to each vertex B of weight w-1 the smallest binary integer satisfying the following two conditions:

(a) The least significant bit of $L^{\tilde{f}}_{mn}(B)$ is $\tilde{f}(B)$ if $\tilde{f}(B)\neq *$;

(b) $L^{\tilde{f}}_{mn}(B) \geq L^{\tilde{f}}_{mn}(A)$ for every directed edge $\overrightarrow{AB}$ terminating at B.

<u>Step 3</u> - Repeat Step 2 until $L^{\tilde{f}}_{mn}(0)$ is assigned.

<u>Step 4</u> - The number of bits in $L^{\tilde{f}}_{mn}(0)$ is the minimal number of negative gates required to realize $\tilde{f}$, and the i-th most significant bit of $L^{\tilde{f}}_{mn}(A)$ is $u_i(A)$ for each $A \in C_n$. (The least significant bit of $L^{\tilde{f}}_{mn}(A)$ shows f(A) where f is the completion of $\tilde{f}$ obtained by this algorithm.)

The only difference from Algorithm 3.1 is in Step 1 and the condition (a) of Step 2. It is evident that this algorithm can be applied to completely specified functions also. The validity of this algorithm can be proved in a manner similar to that for Algorithm 3.1 (MNL).

Algorithm 3.2 (MXL) can be similarly modified for incompletely specified functions.

<u>Algorithm 8.2</u> - Algorithm MXL for an incompletely specified functions $\tilde{f}$.

<u>Step 1</u> - If $\tilde{f}(0)=*$, assign $L^{\tilde{f}}_{mx}(0) = 2^{R_{\tilde{f}}} - 1$; otherwise assign $L^{\tilde{f}}_{mx}(0) = 2^{R_{\tilde{f}}} - 2 + \tilde{f}(0)$, where $R_{\tilde{f}}$ is the minimum number of negative gates required for the realization of $\tilde{f}$ and may be obtained by applying algorithm MNL.

Step 2 - When $L_{mx}^{\tilde{f}}(A)$ is assigned to each vertex A of weight w ($0 \le w \le n-1$) of C_n, assign as $L_{mx}^{\tilde{f}}(B)$ to each vertex B of weight w+1 the largest binary integer satisfying the following two conditions:

(a) The least significant bit of $L_{mx}^{\tilde{f}}(B)$ is $\tilde{f}(B)$ if $\tilde{f}(B) \neq *$;

(b) $L_{mx}^{\tilde{f}}(B) \le L_{mx}^{\tilde{f}}(A)$ for every directed edge $\vec{BA}$ originating from B.

Step 3 - Repeat Step 2 until $L_{mx}^{\tilde{f}}(I)$ is assigned.

Step 4 - Let the i-th most significant bit of $L_{mx}^{\tilde{f}}(A)$ be $u_i(A)$ for every $A \in C_n$. The resulting $\{u_1,\ldots,u_{R_f}\}$ is an NFS(R_f,$\tilde{f}$), where $f = u_{R_{\tilde{f}}}$ is the negative completion of $\tilde{f}$ obtained by this algorithm.

In order to modify Algorithm 6.4.1 to be applicable to incompletely specified functions, we have to first modify Definition 6.3.1 as follows.

Definition 8.1 - A partially specified NFS of length R(=$R_{\tilde{f}}$) and degree i for an incompletely specified function $\tilde{f}$ is a sequence of R functions denoted by $NFS^i(R,\tilde{f}) = (u_1,\ldots, u_i, u_{i+1}^*,\ldots,u_{R-1}^*, \tilde{u}_R=\tilde{f})$ such that

(i) $u_1,\ldots,u_i$ are completely specified functions with respects to $x_1,\ldots,x_n$;

(ii) $(u_1,\ldots,u_i)$ is an NFS(i,u_i);

(iii) $u_{i+1}^*,\ldots,u_{R-1}^*$ are unspecified functions; and

(iv) $\tilde{u}_R = \tilde{f}$ is an incompletely specified function with respect to $x_1,\ldots,x_n$.

An $NFS^i(R,\tilde{f})$ is said to be <u>feasible</u> if there exists at least one complete specification $(u_{i+1},\ldots,u_{R-1}, f)$ of $(u^*_{i+1},\ldots,u^*_{R-1}, \tilde{f})$ with respect to $x_1,\ldots,x_n$ (i.e., a completion[†] of $NFS^i(R,\tilde{f})$) such that $(u_1,\ldots,u_{R-1}, f)$ is an NFS(R,f); if there exists no such completion, $NFS^i(R,\tilde{f})$ is called an <u>infeasible</u> partially specified NFS.

The only difference between Definition 6.3.1 and Definition 8.1 is that the latter contains the last function in the partially specified NFS as an incompletely specified function. The definition of feasibility also takes into consideration the possible completions of $\tilde{f}$.

Based on the modified definition, algorithms CMNL (Algorithm 6.3.1) and CMXL (Algorithm 6.3.2) can be modified accordingly.

<u>Algorithm 8.3</u> - Algorithm CMNL for a feasible $NFS^{i-1}(R,\tilde{f}) = u_1,\ldots,u_{i-1}, u^*_i,\ldots,u^*_{R-1},\tilde{f})$ where $\tilde{f}$ is an incompletely specified function of $x_1,\ldots,x_n$.

The conditional minimum labeling of $NFS^{i-1}(R,\tilde{f})$ $(R=R_{\tilde{f}})$ is a completion of $NFS^{i-1}(R,\tilde{f})$ denoted by $\underline{NFS}^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, \underline{u}_i,\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$ such that in the corresponding labeled n-cube, $C_n(u_1,\ldots,u_{i-1}, \underline{u}_i,\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$, the label $\ell(A; u_1,\ldots,u_{i-1}, \underline{u}_i,\ldots, \underline{u}_{R-1}, \underline{\tilde{f}}) \equiv L^{\tilde{f},i}_{mn}(A)$ takes a minimum possible value for every $A \in C_n$.

[†]Notice that the completion here includes the completion of $\tilde{f}$ also.

<u>Step 1</u> - If $\tilde{f}(I) = *$, assign $L_{mn}^{\tilde{f},i-1}(I) = \sum_{k=0}^{i-1} 2^{R-k} u_k(I)$; otherwise assign $L_{mn}^{\tilde{f},i-1}(I) = \sum_{k=1}^{i-1} 2^{R-k} u_k(I) + \tilde{f}(I)$.

<u>Step 2</u> - When $L_{mn}^{\tilde{f},i-1}(A)$ is assigned to each vertex A of weight $w(1 \leq w \leq n)$ of C_n, assign as $L_{mn}^{\tilde{f},i-1}(B)$ to each vertex B of weight w-1 the smallest binary integer satisfying the following three conditions:

(a) The k-th most significant bit of $L_{mn}^{\tilde{f},i-1}(B)$ is $u_k(B)$, for k=1,...,i-1;

(b) If $\tilde{f}(B) \neq *$, the least significant bit of $L_{mn}^{\tilde{f},i-1}(B)$ is $\tilde{f}(B)$;

(c) $L_{mn}^{\tilde{f},i-1}(B) \geq L_{mn}^{\tilde{f},i-1}(A)$ for every directed edge $\vec{AB}$ terminating at B.

<u>Step 3</u> - Repeat Step 2 until $L_{mn}^{\tilde{f},i-1}(O)$ is assigned.

<u>Step 4</u> - The k-th most significant bit of $L_{mn}^{\tilde{f},i-1}(A)$ is denoted by $\underline{u}_k(A)$ for k=1,...,R-1, and the least significant bit of $L_{mn}^{\tilde{f},i-1}(A)$ is denoted by $\underline{\tilde{f}}(A)$ for each $A \in C_n$. $\underline{NFS}^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, \underline{u}_1,\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$ has been obtained as the completion of $NFS^{i-1}(R,\tilde{f})$ by CMNL.

The modified algorithm CMNL differs from the original one only in Step 1 and condition (b) of Step 2. Similarly, algorithm CMXL (Algorithm 6.2) can be modified as follows.

<u>Algorithm 8.4</u> - Algorithm CMXL for a feasible $NFS^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, u_i^*,\ldots,u_{R-1}^*,\tilde{f})$ where $\tilde{f}$ is an incompletely specified function of n variables.

The conditional maximum labeling of $NFS^{i-1}(R,\tilde{f})$ is a completion of $NFS^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, u_i^*,\ldots,u_{R-1}^*,\tilde{f})$ denoted by $\widehat{NFS}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, \hat{u}_i,\ldots,\hat{u}_{R-1},\hat{\tilde{f}})$ such that, in the corresponding labeled n-cube $C_n(u_1,\ldots,u_{i-1}, u_i,\ldots,u_{R-1},f)$, the label $\ell(A; u_1,\ldots,u_{i-1}, \hat{u}_i,\ldots,\hat{u}_{R-1},\hat{\tilde{f}}) \equiv L_{mx}^{\tilde{f},i-1}(A)$ takes the maximal possible value for each $A \in C_n$.

<u>Step 1</u> - If $\tilde{f}(0) = *$, assign as $L_{mx}^{\tilde{f},i-1}(0)$ the value $\sum_{k=1}^{i-1} 2^{R-k} u_k(0) + \sum_{k=i}^{R} 2^{R-k}$; otherwise assign as $L_{mx}^{\tilde{f},i-1}(0)$ the value $\sum_{k=1}^{i-1} 2^{R-k} u_k(0) + \sum_{k=i}^{R-1} 2^{R-k} + \tilde{f}(0)$.

<u>Step 2</u> - When $L_{mx}^{\tilde{f},i-1}(A)$ is assigned to each vertex A of weight $w(0 \leq w \leq n-1)$ of C_n, assign as $L_{mx}^{\tilde{f},i-1}(B)$ to each B of weight w+1 the largest binary integer satisfying the following conditions:

(a) The k-th most significant bit of $L_{mx}^{\tilde{f},i-1}(B)$ is $u_k(B)$ for $k=1,\ldots,i-1$;

(b) If $\tilde{f}(B) \neq *$, the least significant bit of $L_{mx}^{\tilde{f},i-1}(B)$ is $\tilde{f}(B)$;

(c) $L_{mx}^{\tilde{f},i-1}(B) \leq L_{mx}^{\tilde{f},i-1}(A)$ for each directed edge $\vec{BA}$ originating from B.

<u>Step 3</u> - Repeat Step 2 until $L_{mx}^{\tilde{f},i-1}(I)$ is assigned.

Step 4 - The k-th most significant bit of $L_{mx}^{\tilde{f},i-1}(A)$ is denoted by $\hat{u}_k(A)$ for $k=i,\ldots,R-1$, and the least significant bit of $L_{mx}^{\tilde{f},i-1}(A)$ is denoted by $\hat{\tilde{f}}(A)$ for every $A \varepsilon C_n$.

Like the modifications of algorithm CMXL, only Step 1 and condition (b) of Step 2 have been slightly changed. The discussion about the validity of the algorithms following Algorithms 6.3.1 and 6.3.2 is still valid.

Algorithm 6.3.3 (MPF) need not be changed except for some notations.

Algorithm 8.5 - Algorithm MPF to obtain $\tilde{u}_i$ for a given feasible $NFS^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, u_i^*,\ldots,u_{R-1}^*,\tilde{f})$.

Step 1 - Obtain a completion $\underline{NFS}^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, \underline{u}_i,\ldots, \underline{u}_{R-1}, \underline{\tilde{f}})$ of $NFS^{i-1}(R,f)$ according to Algorithm 8.3 (CMNL).

Step 2 - Obtain another completion $\widehat{NFS}^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, \hat{u}_i,\ldots,\hat{u}_{R-1}, \hat{\tilde{f}})$ of $NFS^{i-1}(R,f)$ according to Algorithm 8.4 (CMXL).

Step 3 - For each vertex A of C_n do the following:

(i) assign $\tilde{u}_i(A) = 0$ if and only if $\underline{u}_i(A)=0$ and $\hat{u}_i(A)=0$;

(ii) assign $\tilde{u}_i(A) = 1$ if and only if $\underline{u}_i(A)=1$ and $\hat{u}_i(A)=1$;

(iii) assign $\tilde{u}_i(A) = *$ (don't care) if and only if $\underline{u}_i(A)=0$ and $\tilde{u}_i(A)=1$.

Algorithm 8.5, the modification of Algorithm 6.3.3 (MPF) does not affect the properties of $\tilde{u}_i$ discussed in Section 6.3. Algorithm 6.4.1 (DIMN) is based on Algorithms 6.3.1 (CMNL), 6.3.2 (CMXL), and 6.3.3 (MPF), and need not be modified except for some symbols related to f, as follows.

<u>Algorithm 8.6</u> - Algorithm DIMN for an incompletely specified function $\tilde{f}$.

<u>Step 1</u> - Let $NFS^{o}(R,\tilde{f}) = (u_1^*,\ldots,u_{R-1}^*,\tilde{f})$ and set $i=1$. ($R \equiv R_{\tilde{f}}$ is the minimum number of cells required for the realization of $\tilde{f}$. If it is not known, it will be determined after $\underline{NFS}^{o}(R,f)$ is obtained in Step 2 by applying Algorithm 8.1 (MNL).)

<u>Step 2</u> - Apply Algorithm 8.3 (CMNL) to obtain $\underline{NFS}^{i-1}(u_1,\ldots,u_{i-1}, \underline{u}_i,\ldots,\underline{u}_{R-1}, \underline{\tilde{f}})$.

<u>Step 3</u> - Apply Algorithm 8.4 (CMXL) to obtain $\widehat{NFS}^{i-1}(R,\tilde{f}) = u_1,\ldots,u_{i-1}, \hat{u}_i,\ldots,\hat{u}_{R-1}, \hat{\tilde{f}})$.

<u>Step 4</u> - Obtain function $\tilde{u}_i$ satisfying:

$\tilde{u}_i(A)=0$, if $\underline{u}_i(A) = \hat{u}_i(A)=0$;

$\tilde{u}_i(A)=1$, if $\underline{u}_i(A) = \hat{u}_i(A)=1$;

$\tilde{u}_i(A)=*$, if $\underline{u}_i(A)=0$ and $\hat{u}_i(A)=1$.

($\underline{u}_i(A)=1$ and $\hat{u}_i(A)=0$ can never occur.)

<u>Step 5</u> - Obtain an irredundant MOS cell configuration for $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ by an appropriate algorithm (e.g., Algorithm 5.1 or 5.2). Let u_i denote the function realized by this cell. (u_i is a completion of $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$.)

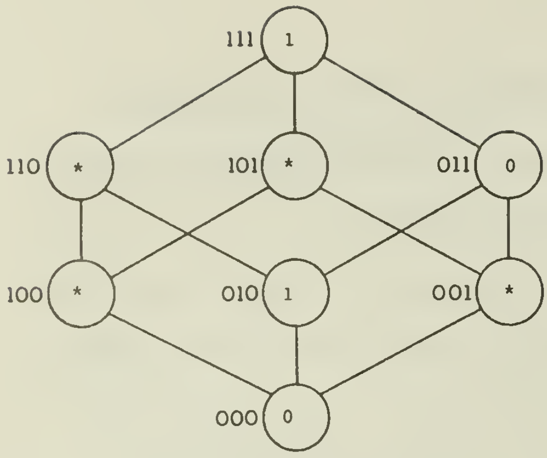
<u>Step 6</u> - If $i = R\text{-}1$, design an irredundant MOS cell configuration of $\tilde{f}$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{R-1}$ and terminate this algorithm; otherwise set $i = i+1$ and go to Step 2.

Since $\tilde{f}$ is an incompletely specified function, after Step 5 of the (R-1)-st iteration, we will have $NFS^{R-1}(R,\tilde{f}) = (u_1,\ldots,u_{R-1}, \tilde{f})$, where $\tilde{f}$ is still incompletely specified with respect to $x_1,\ldots,x_n$. We need not apply Steps 2, 3, and 4 to obtain $(u_1,\ldots,u_{R-1}, \underline{\tilde{f}})$, $(u_1,\ldots,u_{R-1}, \hat{\tilde{f}})$, and $(u_1,\ldots,u_{R-1}, \hat{\tilde{f}})$, respectively, since Step 6 will design an irredundant MOS cell configuration for $\tilde{f}$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{R-1}$, which determines the function f (a completion of $\tilde{f}$) realized by the MOS network.

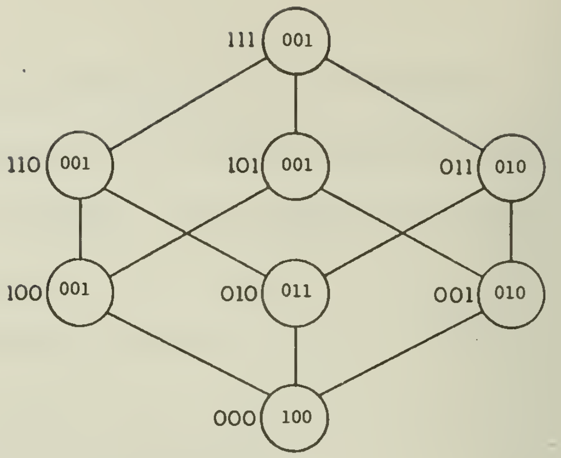
It is not difficult to understand that Theorem 6.4.1, which proves that an MOS network designed by Algorithm 6.4.1 (DIMN) for a function f is irredundant, is still valid even if the function is an incompletely specified function $\tilde{f}$.

It should be noted that an infeasible partially specified $NFS^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, \overset{*}{u}_i,\ldots,\overset{*}{u}_{R-1}, \tilde{f})$ means that no completion f of $\tilde{f}$ can be realized by R negative gates with the first i-1 gates realizing $u_1,\ldots,u_{i-1}$. The complete proof is omitted here.

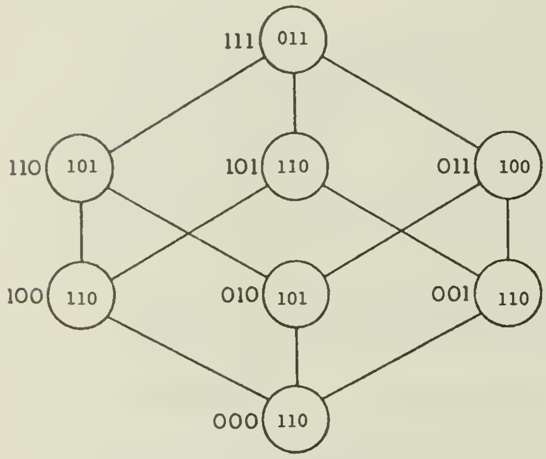
<u>Example 8.1</u> - Consider the incompletely specified function $\tilde{f}$ shown in Fig. 8.1(a) in the form of $C_3(\tilde{f})$, where only $\tilde{f}(111)=1$, $\tilde{f}(011)=0$, $\tilde{f}(010)=1$, and $\tilde{f}(000)=0$ are specified. Step 2, Step 3, and Step 4 of Algorithm 8.6 obtain $\underline{NFS}^{0}(3,\tilde{f})$, $\widehat{NFS}^{0}(3,\tilde{f})$, and $\widetilde{NFS}^{0}(3,\tilde{f})$ as shown in Fig. 8.1(b),(c), and (d), respectively. For $\tilde{u}_1$, three irredundant



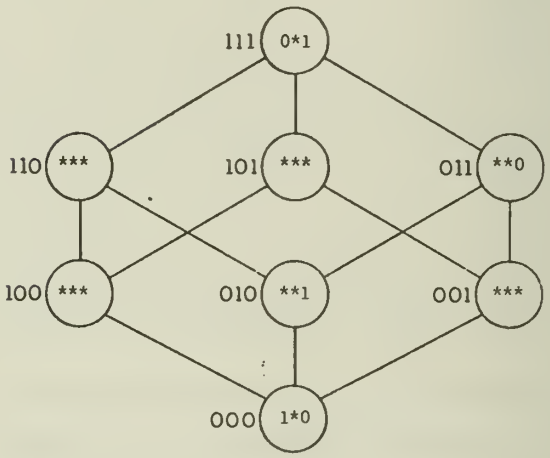
(a) Incompletely specified function $\tilde{f}$.



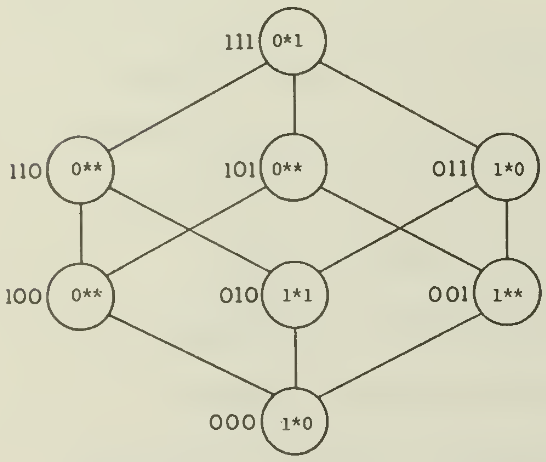
(b) $\underline{NFS}^{o}(3,\tilde{f}) = (\underline{u}_1, \underline{u}_2, \underline{\tilde{f}})$.



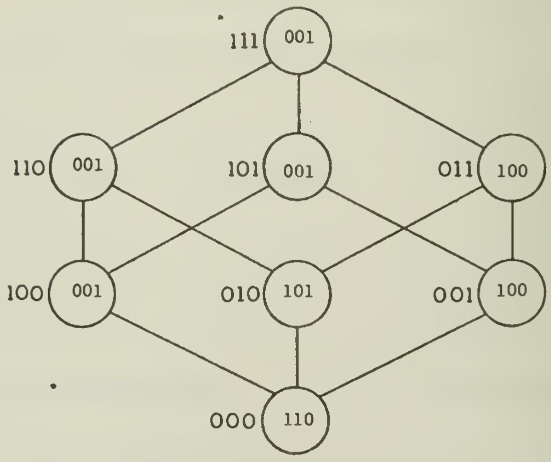
(c) $\hat{NFS}^{o}(3,\tilde{f}) = (\hat{u}_1, \hat{u}_2, \hat{\tilde{f}})$.



(d) $\tilde{NFS}^{o}(3,\tilde{f}) = (\tilde{u}_1, u_2^*, \tilde{f})$.

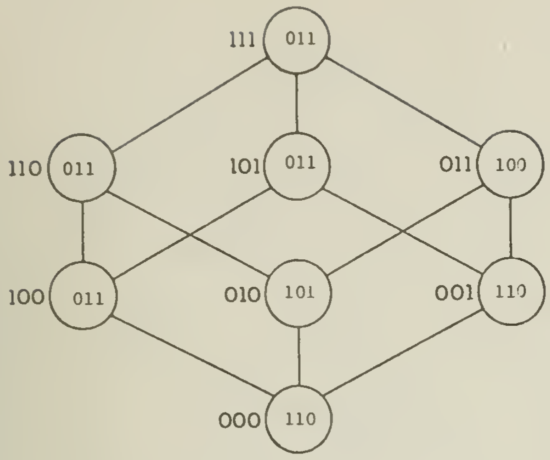


(e) $NFS^{1}(3,\tilde{f})_1 = (u_1, u_2^*, \tilde{f})$ with $u_1 = \bar{x}_1$.

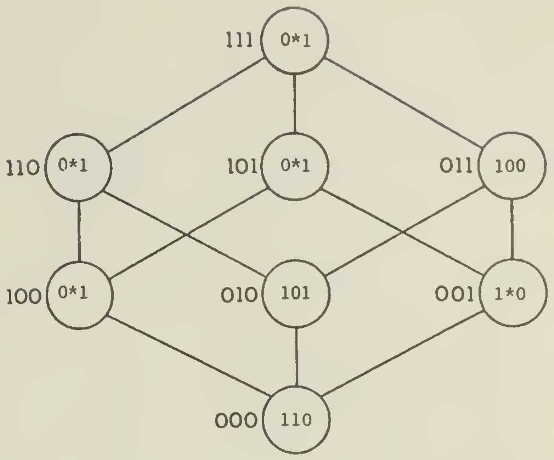


(f) $\underline{NFS}^{1}(3,\tilde{f})_1 = (u_1, \underline{u}_2, \underline{\tilde{f}})$.

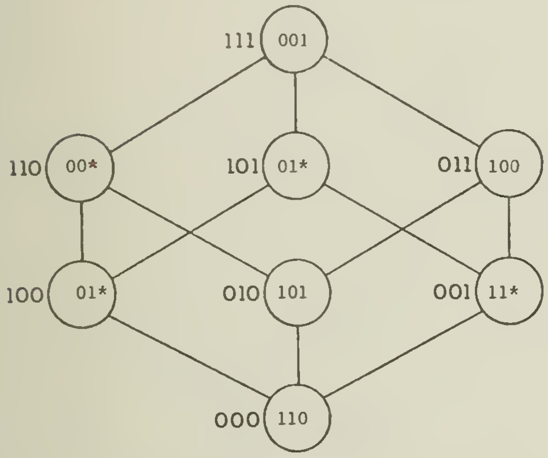
Fig. 8.1 Example 8.1.



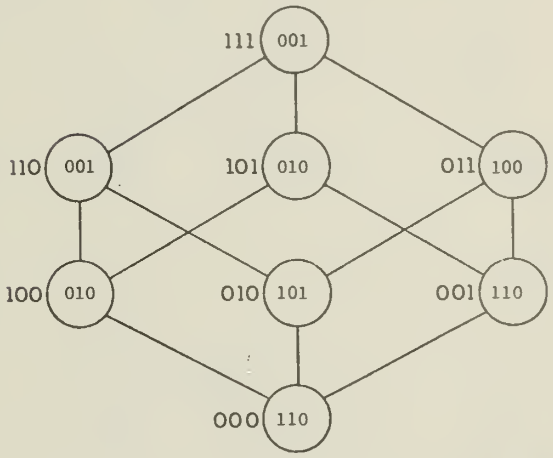
(g) $\widehat{NFS}^1(3,\hat{f})_1 = (u_1, \hat{u}_2, \hat{f})$.



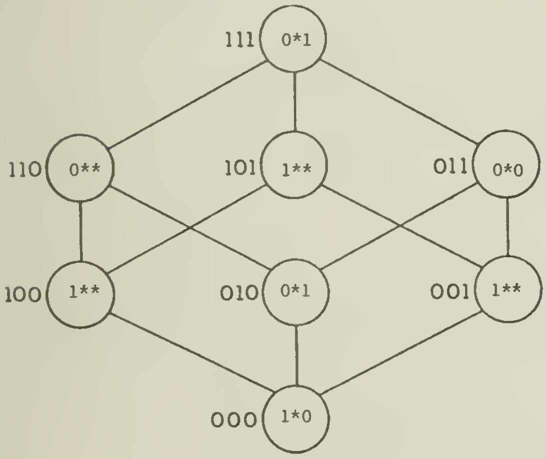
(h) $\widetilde{NFS}^1(3,\hat{f})_1 = (u_1, \tilde{u}_2, \hat{f})$.



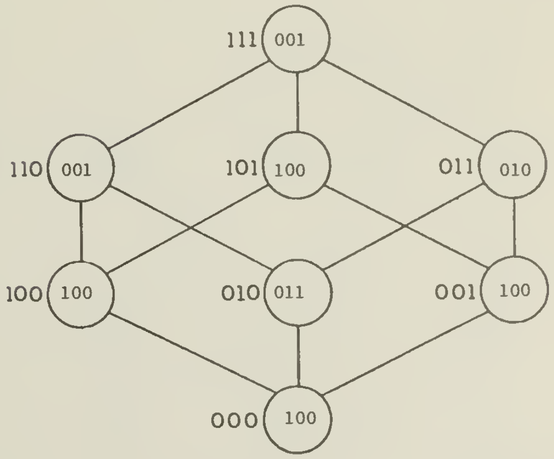
(i) $NFS^2(3,\hat{f})_1 = (u_1, u_2, \hat{f})$.



(j) $NFS(3,\hat{f})_1 = (u_1,u_2,f)$ with $f = \overline{u_2 \vee x_3 u_1}$

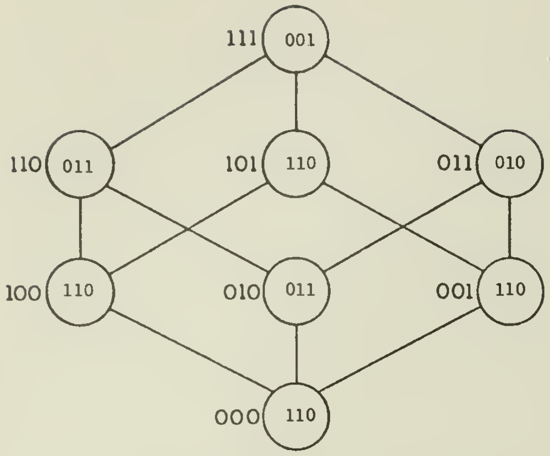


(k) $NFS^1(3,\hat{f})_2 = (u_1,u_2^*,\hat{f})$ with $u_1 = \bar{x}_2$.

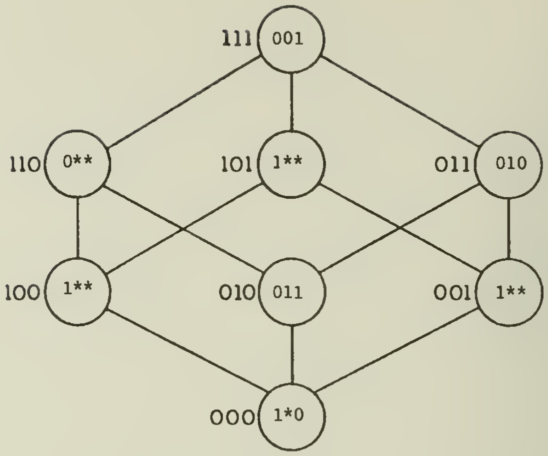


(ℓ) $\underline{NFS}^1(3,\hat{f})_2 = (u_1, \underline{u}_2, \underline{\hat{f}})$.

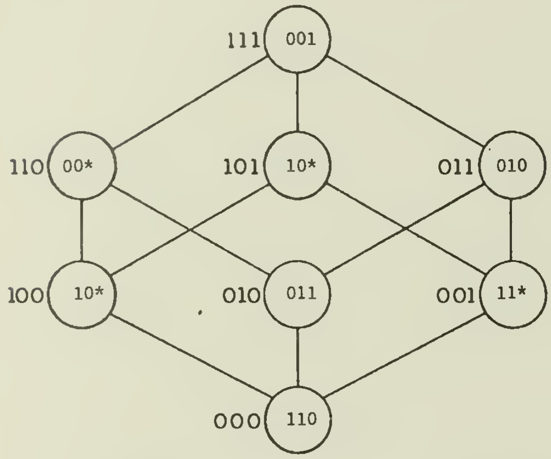
Fig. 8.1 (Continued)



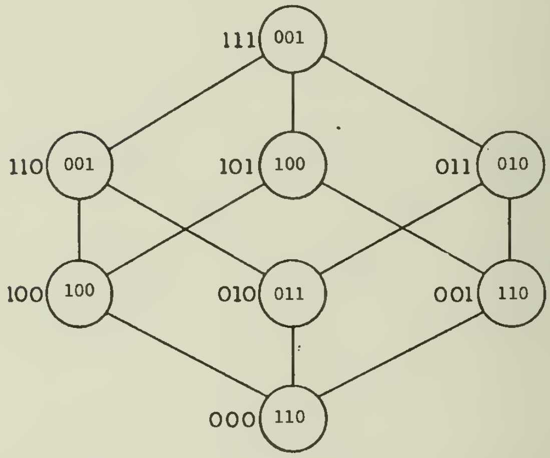
(m) $\widehat{\mathrm{NFS}}^{1}(3,\check{f})_{2} = (u_{1}, \hat{u}_{2}, \check{f})$.



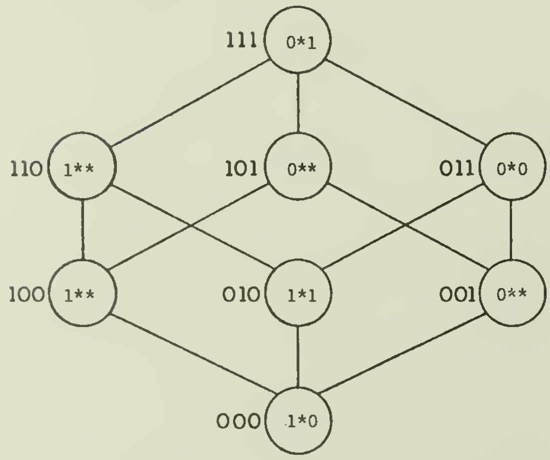
(n) $\mathrm{N\check{F}S}^{1}(3,f)_{2} = (u_{1}, \tilde{u}_{2}, \check{f})$.



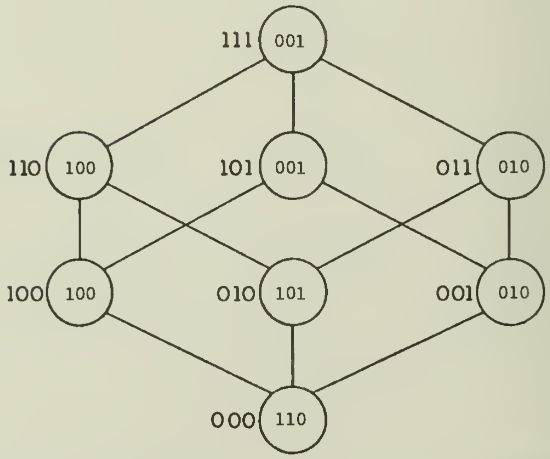
(o) $\mathrm{NFS}^{2}(3,\check{f})_{2} = (u_{1}, u_{2}, \check{f})$.



(p) $\mathrm{NFS}(3,\check{f})_{2} = (u_{1},u_{2},f)$ with $f = \overline{u_{1} \vee x_{3}u_{2}}$

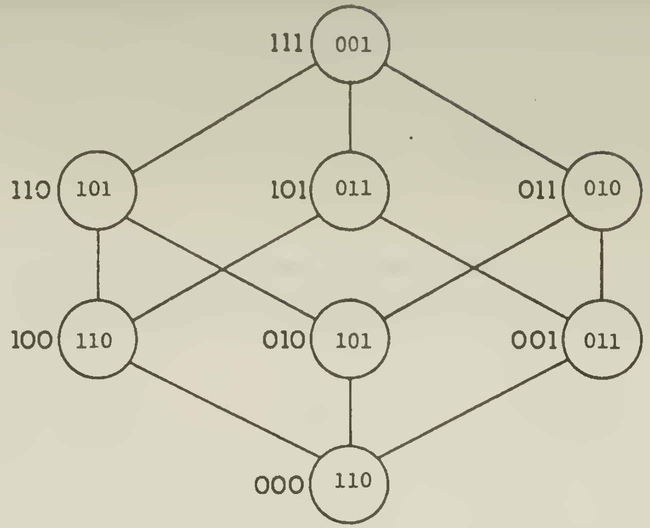


(q) $\mathrm{NFS}^{1}(3,\check{f})_{3} = (u_{1},u_{2}^{*},\check{f})$ with $u_{1} = \bar{x}_{3}$.

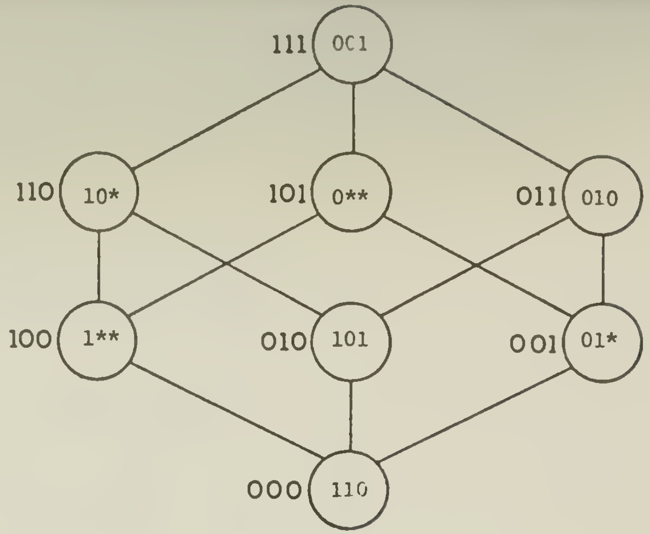


(r) $\underline{\mathrm{NFS}}^{1}(3,\check{f})_{3} = (u_{1}, \underline{u}_{2}, \check{f})$.

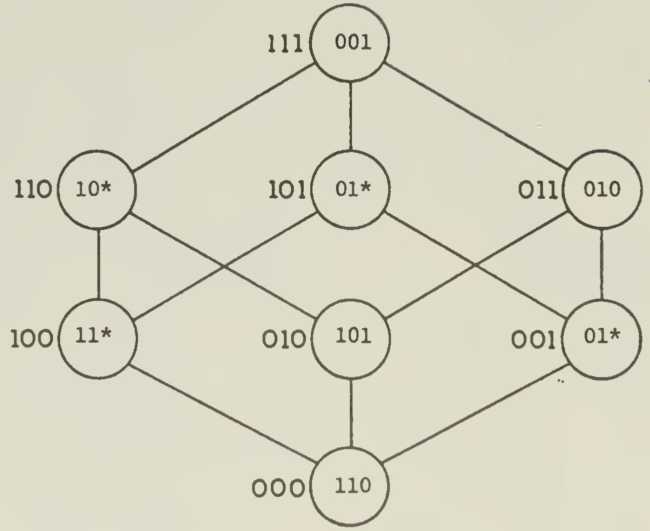
Fig. 8.1 (Continued)



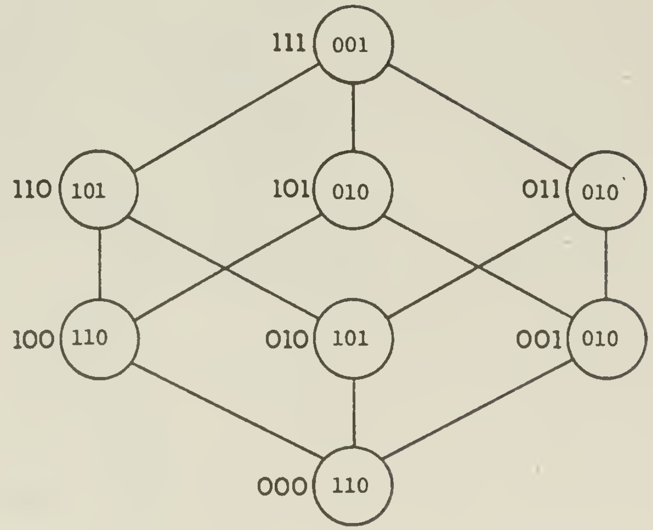
(s) $\widehat{NFS}^1(3,\tilde{f})_3 = (u_1,\hat{u}_2,\hat{\tilde{f}})$.



(t) $\tilde{NFS}^1(3,\tilde{f})_3 = (u_1,\tilde{u}_2,\tilde{f})$

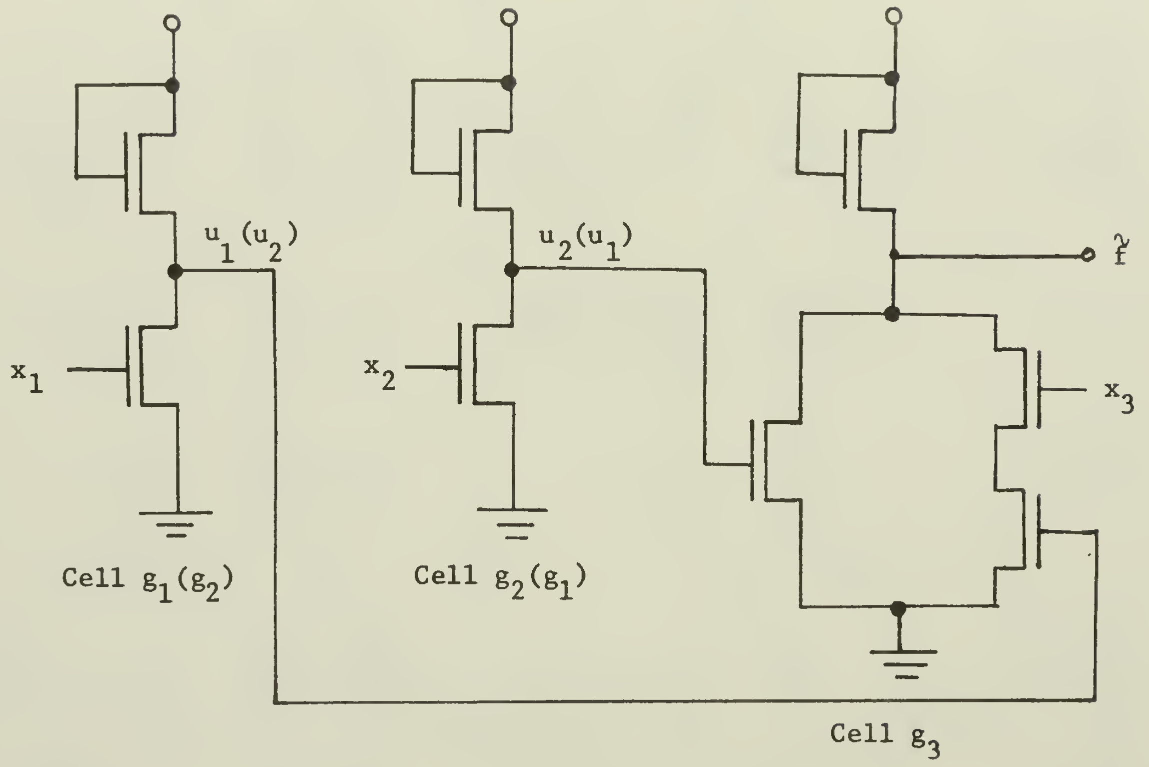


(u) $NFS^2(3,\tilde{f})_3 = (u_1,u_2,\tilde{\tilde{f}})$ with $u_2 = \overline{x_2(x_1 \vee u_1)}$.



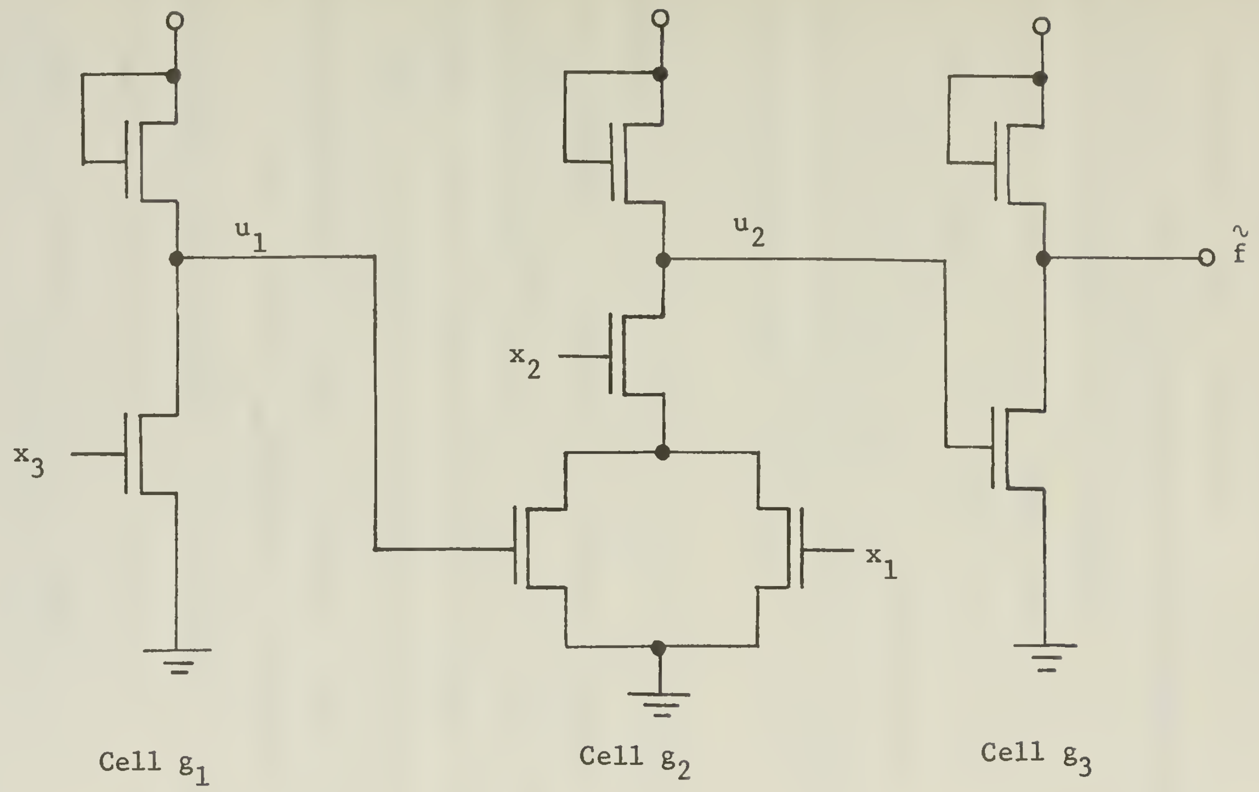
(v) $NFS(3,\tilde{f})_3 = (u_1,u_2,f)$ with $f = \pi_3$.

Fig. 8.1 (Continued)



(w) Irredundant MOS network for $\tilde{f}$ corresponding to NFS$(3,\tilde{f})$, in (j). (If cells g_1 and g_2 are interchanged, this corresponds to NFS$(3,\tilde{f})_2$).

<u>Fig. 8.1</u> (Continued)



(x) Irredundant MOS network for $\tilde{f}$ corresponding to NFS$(3,\tilde{f})_3$ in (v).

<u>Fig. 8.1</u> (Continued)

MOS cell configurations, $u_1 = \bar{x}_1$, $u_1 = \bar{x}_2$, $u_1 = \bar{x}_3$, are obtained. Fig. 8.1(e)-(j) show the subsequent results obtained based on $u_1 = \bar{x}_1$, i.e., $NFS^1(3,\tilde{f})_1$, $\underline{NFS}^1(3,\tilde{f})_1$, $\hat{NFS}^1(3,\tilde{f})_1$, $\tilde{NFS}^1(3,\tilde{f})_1$, $NFS^2(3,\tilde{f})_1$, and $NFS(3,\tilde{f})_1$, respectively. The corresponding irredundant MOS network for $\tilde{f}$ is shown in Fig. 8.1(w). If $u_1 = \bar{x}_2$ is chosen in Step 5 for the first iteration, eventually the same network (cells g_1 and g_2 are interchanged) as that in Fig. 8.1(w) will be obtained as shown by the sequence of Fig. 8.1(k), (l), (m), (n), (o), and (p). On the other hand, if $u_1 = \bar{x}_3$ is chosen as the irredundant MOS cell configuration for $\tilde{u}_1$ in Step 5 for the first iteration, the subsequent intermediate results will become the ones shown in Fig. 8.1 (q), (r), (s), (t), (u), and (v) which lead to the irredundant MOS network for $\tilde{f}$ shown in Fig. 8.1(x). (Not all solutions are shown. For example, from $NFS^1(3,\tilde{f})_3$ in (t) $u_2 = \overline{x_2 \vee x_1 u_1}$ can be obtained.)

Networks designed by algorithm DIMN for incompletely specified functions have diagnostic properties similar to those possessed by the networks designed by Algorithm 6.4.1 DIMN for completely specified functions discussed in Section 7. However, some modifications in the definitions and theorems are necessary.

Corollary 7.1 holds for both completely and incompletely specified functions. The following Corollary 8.1 which corresponds to Corollary 7.1 for incompletely specified functions is based on the properties that the network designed by Algorithm 8.6 (DIMN) for an incompletely specified function $\tilde{f}$ is irredundant with respect to $\tilde{f}$ (i.e., the extraction of any FET from the network will make the resulting network not realizable of any completion of $\tilde{f}$).

Corollary 8.1 - Set $S = \{A \mid \tilde{f}(A) \neq *\}$ is a sufficient test set for all possible single or multiple stuck-at-short and/or stuck-at-open faults in an MOS network designed by Algorithm 8.6 (DIMN) for function $\tilde{f}$.

Definition 8.2 - An inverse pair in $C_n(\tilde{f})$ is an ordered pair of vertices A, B in C_n denoted by $A \rightarrow B$ satisfying:

(i) $A > B$;

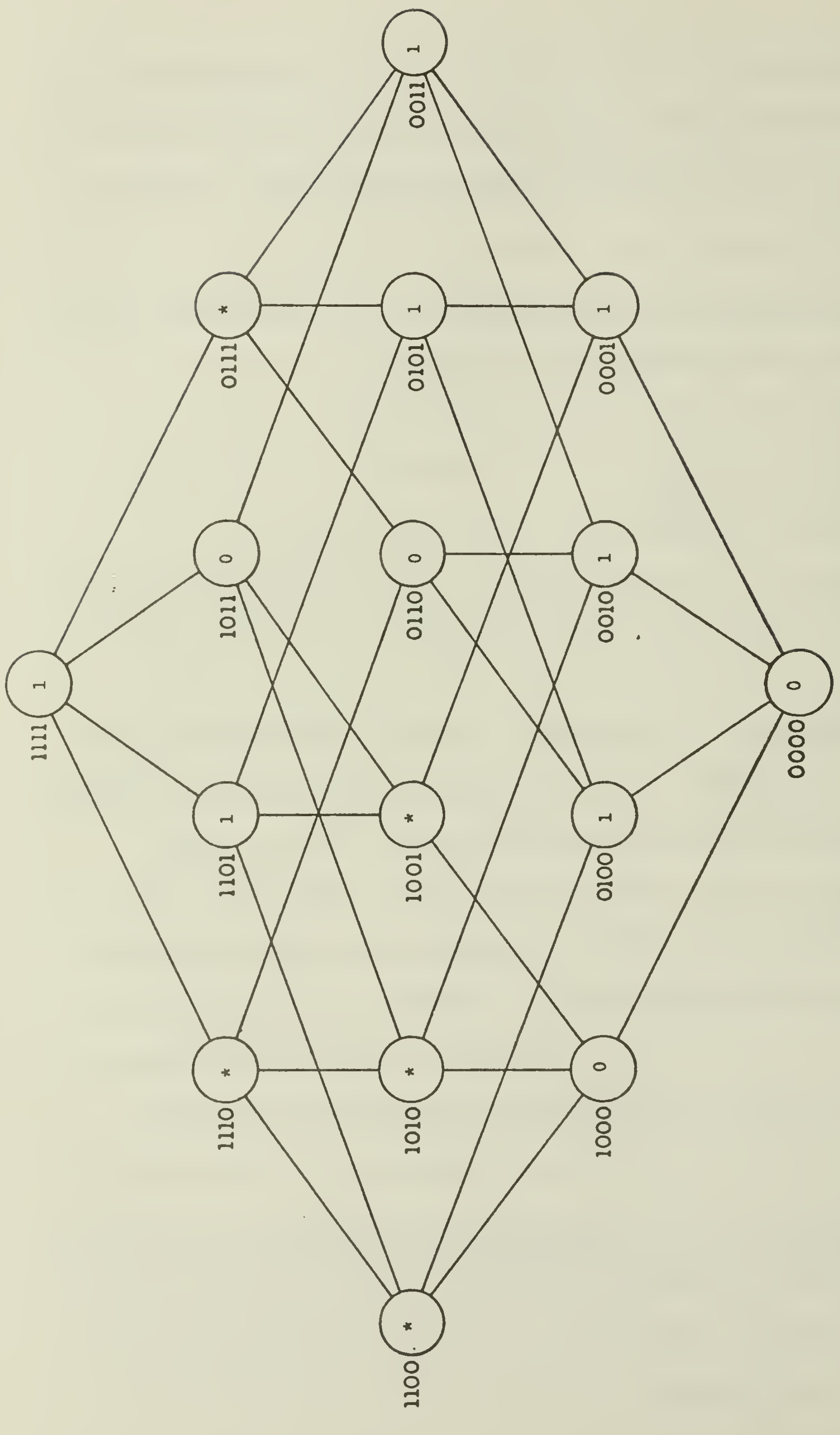
(ii) $\tilde{f}(A)=1$ and $\tilde{f}(B)=0$;

(iii) For every vertex X such that $A > X > B$, $f(X) = *$.

As a special case an inverse edge $\vec{AB}$ is also an inverse pair where no vertex X satisfying $A > X > B$ exists.

Example 8.2 - In the labeled n-cube for the incompletely specified function $\tilde{f}$ shown in the form of $C_4(\tilde{f})$ in Fig. 8.2, the following pairs of specified vectors are inverse pairs: $(1111) \rightarrow (1011)$, $(1111) \rightarrow (0110)$, $(1101) \rightarrow (1000)$, $(0100) \rightarrow (0000)$, $(0010) \rightarrow (0000)$, and $(0001) \rightarrow (0000)$. $(1111) \rightarrow (0110)$ is an inverse pair because $\tilde{f}(1111)=1$, $\tilde{f}(0110)=0$, and $\tilde{f}(1110)=\tilde{f}(0111)=*$. Similarly, $(1101) \rightarrow (1000)$ is an inverse pair because $\tilde{f}(1101)=1$, $\tilde{f}(1000)=0$, and $\tilde{f}(1100)=\tilde{f}(1001)=*$. Each of the other pairs in $C_4(\tilde{f})$ constitutes an inverse edge in $C_4(\tilde{f})$.

Definition 8.3 - The characteristic input set of an incompletely specified function $\tilde{f}$ denoted by $S_C(\tilde{f})$ is the set of vectors each of which is in at least one inverse pair of $C_n(\tilde{f})$ (note that inverse edges are also included).



<u>Fig. 8.2</u> Example 8.2.

Example 8.3 - Consider the function $\tilde{f}$ given in Example 8.2. The characteristic input set of $\tilde{f}$ is

$$S_C(\tilde{f}) = \{(1111), (1011), (1101), (0110), (1000), (0100), (0010), (0001), (0000)\}.$$

Lemma 8.1 - Let $\tilde{f}_1$ and $\tilde{f}_2$ be two incompletely specified functions of n variables such that $\tilde{f}_2(A) = \tilde{f}_1(A)$ for every $A \in S_C(\tilde{f}_1)$. Let the sequence of functions $(u_1,\ldots,u_{R-1}, f_2)$ be a negative function sequence for $\tilde{f}_2$ (i.e., f_2 is a completion of $\tilde{f}_2$), and be denoted by $NFS(R,\tilde{f}_2) = (u_1,\ldots,u_{R-1}, f_2)$. Then $(u_1,\ldots,u_{R-1}, \tilde{f}_1)$ with the same $u_1,\ldots,u_{R-1}$ must be a feasible partially specified NFS of length R and degree R-1.

Proof - Since $(u_1,\ldots,u_{R-1}, f_2)$ is an NFS, $(u_1,\ldots,u_{R-1})$ is an NFS of length R-1. Thus to prove that $(u_1,\ldots,u_{R-1}, \tilde{f}_1)$ is feasible, we only need to prove that $\tilde{f}_1$ is negative with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{R-1}$.

Suppose $\tilde{f}_1$ is not negative with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{R-1}$. Then in the labeled (n+R-1)-cube $C_{n+R-1}(\tilde{f}_1)$ with respect to $\tilde{f}_1$, there must exist at least one pair of vertices, $(a_1,\ldots,a_n, u_1(A),\ldots,u_{R-1}(A))$ and $(b_1,\ldots,b_n, u_1(B),\ldots,u_{R-1}(B))$, such that $(a_1,\ldots,a_n, u_1(A),\ldots,u_{R-1}(A)) > (b_1,\ldots,b_n, u_1(B),\ldots,u_{R-1}(B))$, $\tilde{f}_1(A)=1$, and $\tilde{f}_1(B)=0$. This means that there exists at least one pair of vectors $A, B \in V_n$ such that $\tilde{f}_1(A)=1$, $\tilde{f}_1(B)=0$, $A>B$, and $u_i(A) \geq u_i(B)$ for $i=1,\ldots,R-1$. We will consider all cases with $\tilde{f}_1(A)=1$, $\tilde{f}_1(B)=0$, and $A>B$ to show that $u_i(A) \geq u_i(B)$ can not hold for all

$i=1,\ldots,R-1$ (i.e., there exists some i, $1 \leq i \leq R-1$, such that $u_i(A)=0$ and $u_i(B)=1$).

(a) $A, B \in S_C(\tilde{f}_1)$

In this case, $\tilde{f}_2(A) = \tilde{f}_1(A)$ and $\tilde{f}_2(B) = \tilde{f}_1(B)$ must hold by the assumption. Since $(u_1,\ldots,u_{R-1}, f_2)$ is an NFS, f_2 is negative with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{R-1}$, and therefore by Theorem 2.3 there exists some $1 \leq i \leq R-1$ such that $u_i(A)=0$ and $u_i(B)=1$. Thus $u_i(A) \geq u_i(B)$ can not hold for all $i=1,\ldots,R-1$. It should be noted that $\ell(A; u_1,\ldots,u_{R-1}) < \ell(B; u_1,\ldots,u_{R-1})$ must hold for this case. This relation will be used in (b).

(b) $A \notin S_C(\tilde{f}_1)$ and/or $B \notin S_C(\tilde{f}_1)$

From $\tilde{f}_1(A)=1$, $\tilde{f}_1(B)=0$, $A>B$, and Definitions 8.2 and 8.3, it is obvious that there must exist a pair of vectors $X, Y \in V_n$ such that $X, Y \in S_C(\tilde{f}_1)$, $A \geq X > Y \geq B$, and $\tilde{f}_1(X)=1$, $\tilde{f}_1(Y)=0$. From (a), $\ell(X; u_1,\ldots,u_{R-1}) < \ell(Y; u_1,\ldots,u_{R-1})$ must hold. Since $(u_1,\ldots,u_{R-1})$ is an NFS$(R-1, u_{R-1})$, $\ell(A; u_1,\ldots,u_{R-1}) \leq \ell(X; u_1,\ldots,u_{R-1}) < \ell(Y; u_1,\ldots,u_{R-1}) \leq \ell(B; u_1,\ldots,u_{R-1})$ must hold. From the relation that $\ell(A; u_1,\ldots,u_{R-1}) < \ell(B; u_1,\ldots,u_{R-1})$, it is evident that there exists some $1 \leq i \leq R-1$ such that $u_i(A)=0$ and $u_i(B)=1$.

Consequently, $\tilde{f}_1$ is negative with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{R-1}$, i.e., $(u_1,\ldots,u_{R-1}, \tilde{f}_1)$ is a feasible NFS$^{R-1}(R,\tilde{f}_1)$.

Q.E.D.

Based on this lemma, the following theorem corresponding to Theorem 7.2 for incompletely specified functions can be proved.

Theorem 8.1 - The characteristic input set $S_C(\tilde{f})$ for an incompletely specified function $\tilde{f}$ is a sufficient test set for the set Φ_n of all possible non-pure output cell faults for a network N designed by Algorithm 8.6 (DIMN) for $\tilde{f}$.

Proof - Suppose $S_C(\tilde{f})$ is not a sufficient test set for Φ_n of N. This means that some fault $F \varepsilon \Phi_n$ for N can not be detected by input vectors in $S_C(\tilde{f})$. In other words,

$$f(N(F),A) = f(N,A) \equiv \tilde{f}(A) \quad \text{for every } A \varepsilon S_C(\tilde{f}). \qquad (8.1)$$

Consider the function $\tilde{f}_1$ defined by $\tilde{f}_1(A) = f(N(F),A)$ for every A such that $\tilde{f}(A) \neq *$; and $\tilde{f}_1(A) = *$ for every A such that $\tilde{f}(A) = *$. By the above definition of $\tilde{f}_1$ and (8.1), $S_C(\tilde{f}) \subseteq S_C(\tilde{f}_1)$ must hold. Now let cell g_i be the cell with the smallest i that has faulty FETs in N(F). Because of $F \varepsilon \Phi_n$, $i \neq R$ must hold where $R = R_{\tilde{f}}$. Let the negative function sequence realized by N be denoted by $(u_1,\ldots,u_{R-1}, f(N))$ where f(N) is a completion of $\tilde{f}$. Then the negative function sequence realized by the faulty network N(F) can be written as $(u_1,\ldots,u_{i-1}, \overset{o}{u}_{i+1},\ldots,\overset{o}{u}_{R-1}, f(N(F)))$ where $(u_1,\ldots,u_{i-1})$ are identical to those in the NFS realized by N since there is no faulty FETs in $g_1,\ldots,g_{i-1}$. Due to the manner of defining $\tilde{f}_1$, f(N(F)) is a completion of $\tilde{f}_1$. Now let us compare $(u_1,\ldots,u_{i-1}, \overset{o}{u}_i,\ldots,\overset{o}{u}_{R-1}, f(N(F)))$ and $(u_1,\ldots,u_{i-1}, \overset{o}{u}_i,\ldots,\overset{o}{u}_{R-1}, \tilde{f})$. Then by Lemma 8.1 $(u_1,\ldots,u_{i-1}, \overset{o}{u}_i,\ldots,\overset{o}{u}_{R-1}, \tilde{f})$ must be a feasible $NFS^{R-1}(R,\tilde{f})$ because $\tilde{f}_1(A) = \tilde{f}(A)$ for every $A \varepsilon S_C(\tilde{f})$.

However, since N is designed by Algorithm 8.6 (DIMN), cell g_i is irredundant with respect to $\tilde{u}_i$ (Step 5). Therefore, $\overset{o}{u}_i$ is not a negative completion of $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{R-1}$, where $\tilde{u}_i$ is the maximal permissible function for u_i^* in $NFS^{i-1}(R,\tilde{f}) = (u_1,\ldots,u_{i-1}, u_i^*,\ldots,u_{R-1}^*, \tilde{f})$. Therefore, $(u_1,\ldots,u_{i-1}, \overset{o}{u}_i, u_{i+1}^*,\ldots,u_{R-1}^*, \tilde{f})$ can not be a feasible $NFS^i(R,\tilde{f})$. This contradicts the result shown in the previous paragraph that $(u_1,\ldots,u_{i-1}, \overset{o}{u}_i,\ldots, \overset{o}{u}_{R-1}, \tilde{f})$ is a feasible $NFS^{R-1}(R,\tilde{f})$. Consequently, $S_C(\tilde{f})$ is a sufficient test set for Φ_n of N.

Q.E.D.

Although $S_C(\tilde{f})$ may be a redundant test set for MOS networks designed by Algorithm 8.6 (DIMN) for $\tilde{f}$, $S_C(\tilde{f})$ contains a much smaller number of vectors than V_n when $\tilde{f}$ has a large number of unspecified values (don't cares). Similarly, when $\tilde{f}$ has a large number of unspecified values, the test set given by Corollary 8.1 is relatively small. Since it is a sufficient test set for all possible single and multiple stuck-at-short and/or stuck-at-open faults, it may be used as the test set in practice.

Example 8.4 - Compare the results given in Example 6.5.1 for function $f = x_1x_2 \quad x_2\bar{x}_3$ and Example 8.1 for function $\tilde{f}$ ($\tilde{f}(111)=1$, $\tilde{f}(011)=0$, $\tilde{f}(010)=1$, $\tilde{f}(000)=0$, and $\tilde{f}(A)=*$ for all other $A \in V_3$). Algorithm 8.6 (DIMN) obtains identical results for both functions as shown in Fig. 6.5.1(s), (t), and Fig. 8.1 (w), (x). According to Corollary 8.1, $\Delta = \{(111), (011), (010), (000)\}$ is a sufficient test set for networks shown in Fig. 8.1(w), and (x), and hence for networks shown in Fig. 6.5.1(s) and (t) also.

As mentioned in Section 7, $S_C(f)$ is usually a redundant test set for networks designed by Algorithm 8.6 (DIMN) for function f. Example 8.4 illustrated this fact.

<u>Example 8.5</u> - The networks (not shown) designed by Algorithm 8.6 (DIMN) for the function $\tilde{f}$ given by Fig. 8.2 of Example 8.2 will have a sufficient test set Δ for Φ_n for these networks:

$$\Delta = S_C(\tilde{f}) = \{(1111),(1101),(1011),(0110),(1000),(0100),(0010),(0001),(0000)\}.$$

9. EXTENSION TO MULTIPLE-OUTPUT MOS NETWORKS

This section will discuss the design of irredundant MOS networks for a given set of functions. As discussed in [NTK 72], the problem of designing an MOS network with a minimum number of MOS cells (negative gates) for a set of more than one functions is, in general, a difficult problem. Let $f_1,\ldots,f_m$ be a given set of completely specified functions[†] for which an MOS network with a minimum number of cells is to be designed. The generalized form of such a network is the same as shown in Fig. 3.1 with some gate realizing each function f_j for $j=1,\ldots,m$. Let $p_1,\ldots,p_m$ be a set of numbers which determines the positions of output functions in such a way that $p_i = j$ if and only if the j-th gate (g_j) realizes the i-th function f_i. Unlike the case of single output network, the difficulty of designing a multiple-output network with a minimum number of negative gates lies in determining which $g_i (1 \leq i \leq R)$ is to realize which $f_j (1 \leq j \leq m)$. So let us prefix which gate is to realize which function. Given R and any set of $p_1,\ldots,p_m$,[ʃ] however, it is not difficult to obtain an irredundant MOS networks for $f_1,\ldots,f_m$ with a minimum number of MOS cells under this condition (i.e.,

[†]For the simplicity of discussion, only completely specified functions are considered in this section. The extension to incompletely specified functions is similar to that discussed in Section 8.

[ʃ][NTK 72] presented an example showing how the total number of negative gates can be reduced by properly determining $p_1,\ldots,p_m$. In general, it is not known how to determine $p_1,\ldots,p_m$ so as to minimize the total number of gates. The algorithm presented here, however, can be applied to any set of $p_1,\ldots,p_m$ such as $p_1=R-m+1$, $p_2=R-m+2,\ldots,p_m=R$.

positions of output functions are fixed). The required modifications to algorithm DIMN and its associated algorithms are straightforward. The detailed discussion of the algorithms with modifications required for this case (this will be referred to later as the modified algorithms) is omitted here, but the modifications will be outlined instead along with an illustrative example.

The main difference of the modified algorithm DIMN (for multiple-outputs with fixed output positions) from the original one (Algorithm 6.4.1) is that, in a partially specified negative function sequence in the former algorithm, the function in the p_i-th position of the NFS, for i=1,...,m, is always completely specified, but in the latter case, only the first k functions and the output function are completely specified when the partially specified NFS is of degree k. In other words, in the beginning of the modified algorithm, $NFS^o(R; f_1,\dots,f_m; p_1,\dots,p_m) = (u_1^*,\dots,u_R^*)$ where $u_{p_i}^* = f_i$ for i=1,...,m is completely specified and $NFS^o(R; f_1,\dots,f_m; p_1,\dots,p_m)$ represents the partially specified negative function sequence for output functions $f_1,\dots,f_m$ at output positions $p_1,\dots,p_m$. On the other hand, $NFS^o(R,f) = (u_1^*,\dots,u_{R-1}^*, f)$ for a single function f has only one specified function, f. The conditional minimum labeling or conditional maximum labeling for an $NFS^{i-1}(u_1,\dots,u_{i-1}, u_i^*,\dots,u_{R-1}^*, u_R^*)$ will assign a minimum possible or a maximum possible label L(A) to each vertex $A \varepsilon C_n$ satisfying

(a) The k-th bit of L(A) is $u_k(A)$ for k=1,...,i-1;

(b) The p_k-th bit of L(A) is $f_k(A)$ for k=1,...,m; and

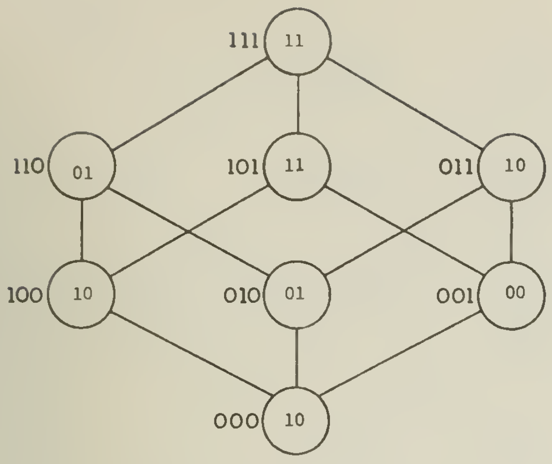
(c) $L(A) \leq L(B)$ for every directed edge $\vec{AB} \varepsilon C_n$.

Algorithm DIMN itself need not be changed except that in the p_k-th iteration for k=1,...,m, Steps 2, 3, and 4 are skipped since $u^*_{p_k} = f_k$ is completely specified with respect to $x_1,\ldots,x_n$.

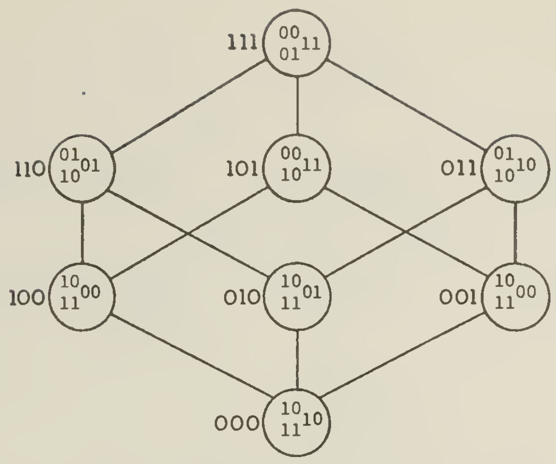
It should be noted that if the given R is too large and the set of $p_1,\ldots,p_m$ is inappropriate, the maximum permissible function u_i for u^*_i of $NFS^{i-1}(R;\ f_1,\ldots,f_m;\ p_1,\ldots,p_m) = (u_1,\ldots,u_{i-1},\ u^*_i,\ldots,u^*_R)$ may have no specified value (i.e., $\tilde{u}_i(A)=*$ for every $A \in C_n$). In such a case, u_i is redundant, so the modified algorithm proceeds to the next iteration to obtain $\tilde{u}_{i+1}$. The designed MOS network for this case will actually contain fewer number of gates than the specified R. It should be noted that the irredundancy of the designed network is still guaranteed.

Corollary 7.1 is still valid for the networks designed by the modified algorithm DIMN for multiple output case. Theorem 7.2 needs minor modification because there are more than one outputs. Let $S_C(f_1,\ldots,f_m) = S_C(f_1)\cup\ldots\cup S_C(f_m)$ be the characteristic set for the set of output functions $f_1,\ldots,f_m$. Then $S_C(f_1,\ldots,f_m)$ is a sufficient test set for all non-pure output cell faults for every network designed by the modified algorithm DIMN for $f_1,\ldots,f_m$.

<u>Example 9.1</u> - Functions f_1 and f_2 are given as shown in Fig. 9.1(a) in the form $C_3(f_1,f_2)$. If f_1 and f_2 are to be realized by gates g_3 and g_4, respectively (since one of the output functions must be realized by the last gate in the network, four is the total number of gates in the network, which is determined by applying the modified minimum labeling algorithm with $f_2(A)$ and $f_1(A)$ being the

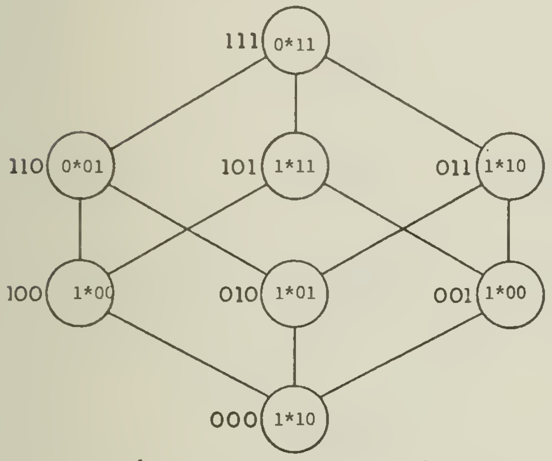


(a) $C_3(f_1,f_2)$.

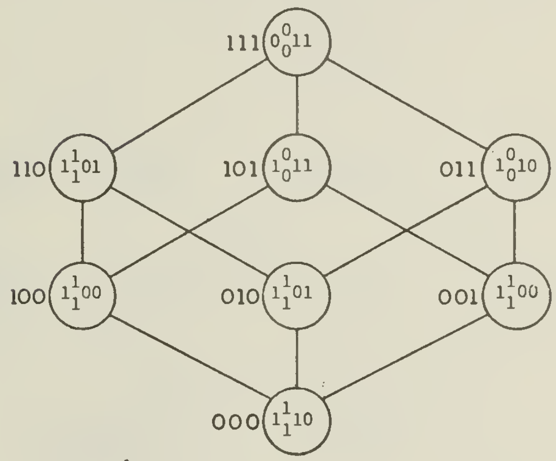


111 $\begin{smallmatrix}00\\01\end{smallmatrix}11$
110 $\begin{smallmatrix}01\\10\end{smallmatrix}01$
101 $\begin{smallmatrix}00\\10\end{smallmatrix}11$
011 $\begin{smallmatrix}01\\10\end{smallmatrix}10$
100 $\begin{smallmatrix}10\\11\end{smallmatrix}00$
010 $\begin{smallmatrix}10\\11\end{smallmatrix}01$
001 $\begin{smallmatrix}10\\11\end{smallmatrix}00$
000 $\begin{smallmatrix}10\\11\end{smallmatrix}10$

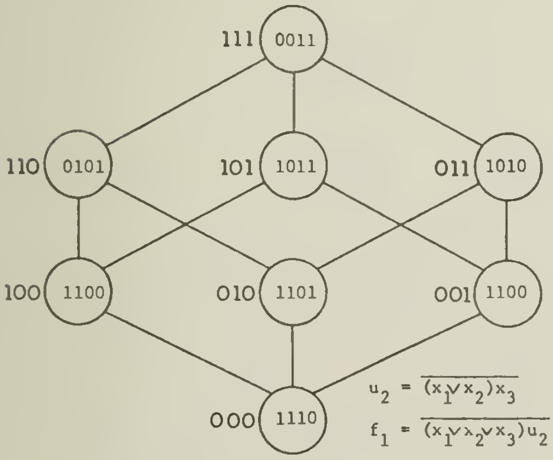
(b) $\underline{NFS}^0(4; f_1,f_2; 3,4) = (\underline{u}_1,\underline{u}_2, f_1,f_2)$.
$\widehat{NFS}^0(4, f_1,f_2; 3,4) = (\hat{u}_1,\hat{u}_2, f_1,f_2)$.



(c) $NFS^1(4; f_1,f_2; 3,4) = (\overline{x_1x_2}, u_2^*, f_1,f_2)$.



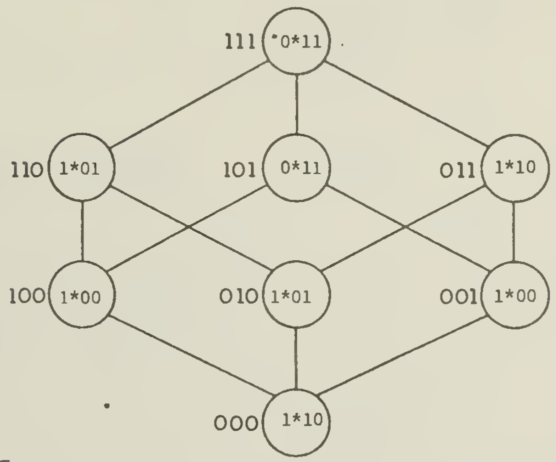
(d) $\underline{NFS}^1(4; f_1,f_2; 3,4)$.
$\widehat{NFS}^1(4; f_1,f_2; 3,4)$.



$u_2 = \overline{(x_1 \vee x_2)x_3}$

$f_1 = \overline{(x_1 \vee x_2 \vee x_3)u_2}$

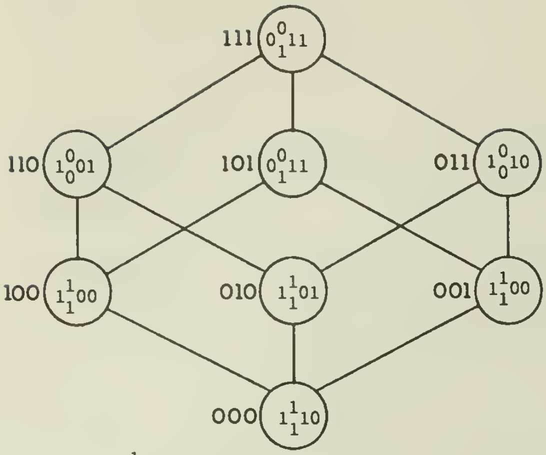
$f_2 = \overline{u_1(u_2(x_1 \vee x_3 \vee f_1) \vee x_2x_3)}$



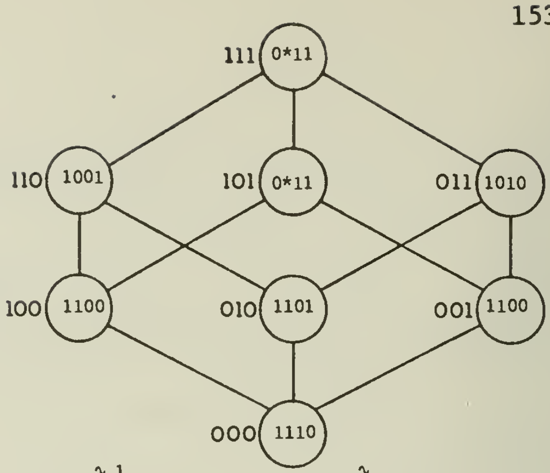
(f) $NFS^1(4; f_1,f_2; 3,4)_2 = (\overline{x_1x_3}, u_2^*, f_1,f_2)$.

(e) $\widehat{NFS}^1(4; f_1,f_2; 3,4) = NFS(4; f_1,f_2; 3,4) = (u_1,u_2, f_1,f_2)$.

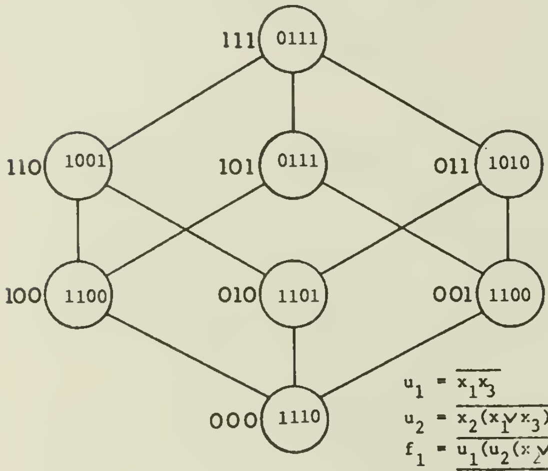
<u>Fig. 9.1</u> Example 9.1.



(g) $\underline{\text{NFS}}^1(4;\ f_1,f_2;\ 3,4)_2 = (u_1,\underline{u}_2,f_1,f_2)$.
$\widehat{\text{NFS}}^1(4;\ f_1,f_2;\ 3,4)_2 = (u_1,\hat{u}_2,f_1,f_2)$.

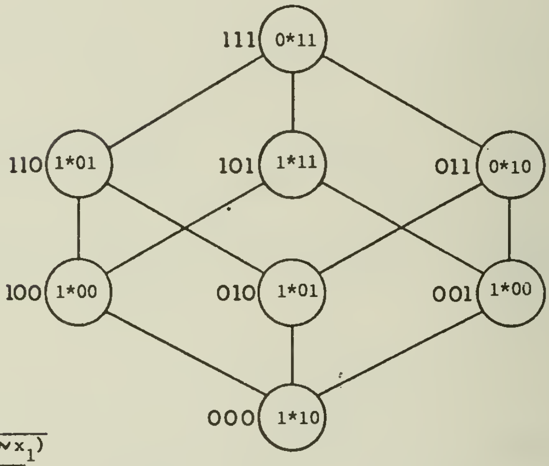


(h) $\widetilde{\text{NFS}}^1(4;\ f_1,f_2;\ 3,4)_2 = (u_1\tilde{u}_2,\ f_1,f_2)$.

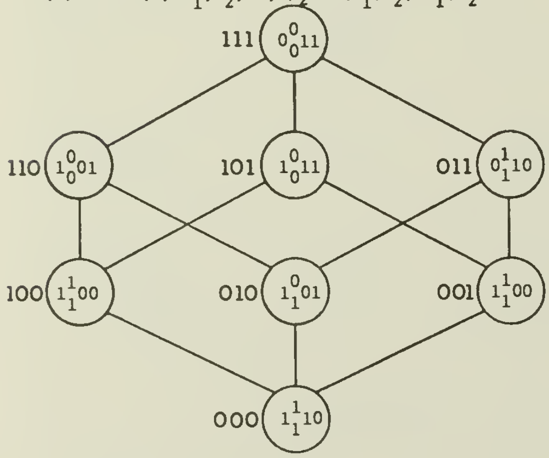


$$u_1 = \overline{x_1x_3}$$
$$u_2 = \overline{x_2(x_1\vee x_3)}$$
$$f_1 = \overline{u_1(u_2(x_2\vee x_3)\vee x_1)}$$
$$f_2 = \overline{u_1(f_1\vee x_3\vee}\,\overline{x_1u_2)}$$

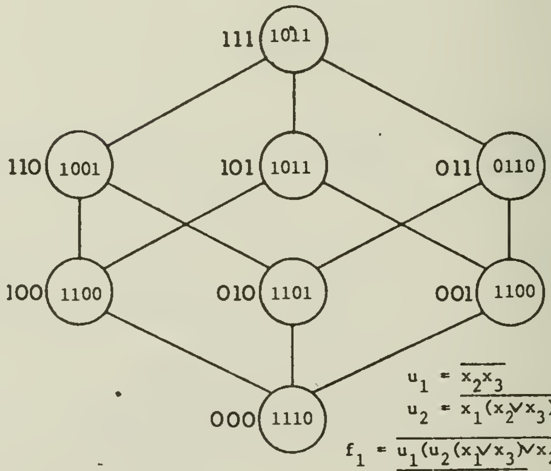
(i) $\text{NFS}^2(4;\ f_1,f_2;\ 3,4)_2 = (u_1,u_2,\ f_1,f_2)$.



(j) $\text{NFS}^1(4;\ f_1,f_2;\ 3,4)_3 = (\overline{x_2x_3},u_2^*,f_1,f_2)$.



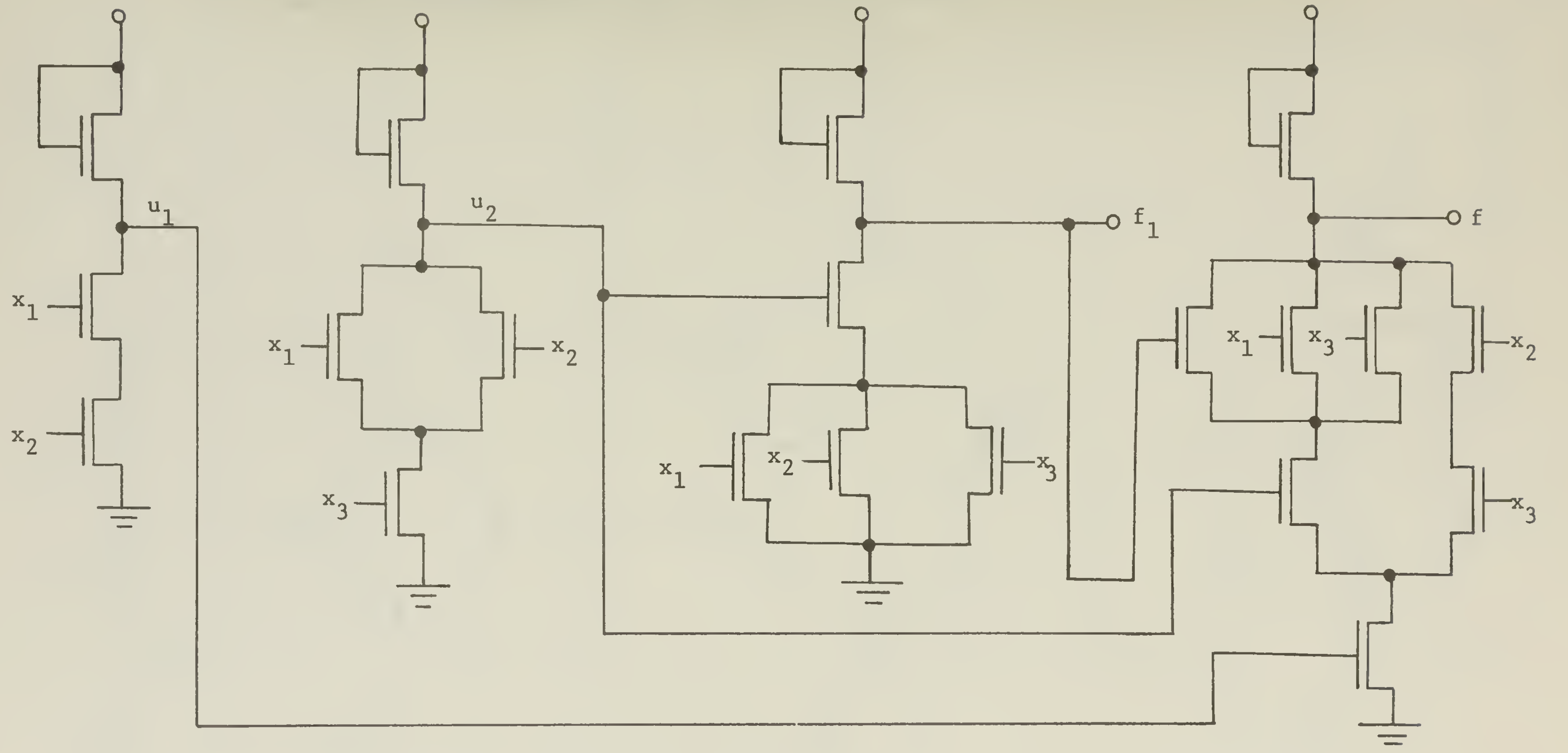
(k) $\underline{\text{NFS}}^1(4;\ f_1,f_2;\ 3,4)_3$.
$\widehat{\text{NFS}}^1(4;\ f_1,f_2;\ 3,4)_3$.



$$u_1 = \overline{x_2x_3}$$
$$u_2 = \overline{x_1(x_2\vee x_3)}$$
$$f_1 = \overline{u_1(u_2(x_1\vee x_3)\vee x_2)}$$
$$f_2 = \overline{u_2(x_1\vee x_3\vee f_1)}$$

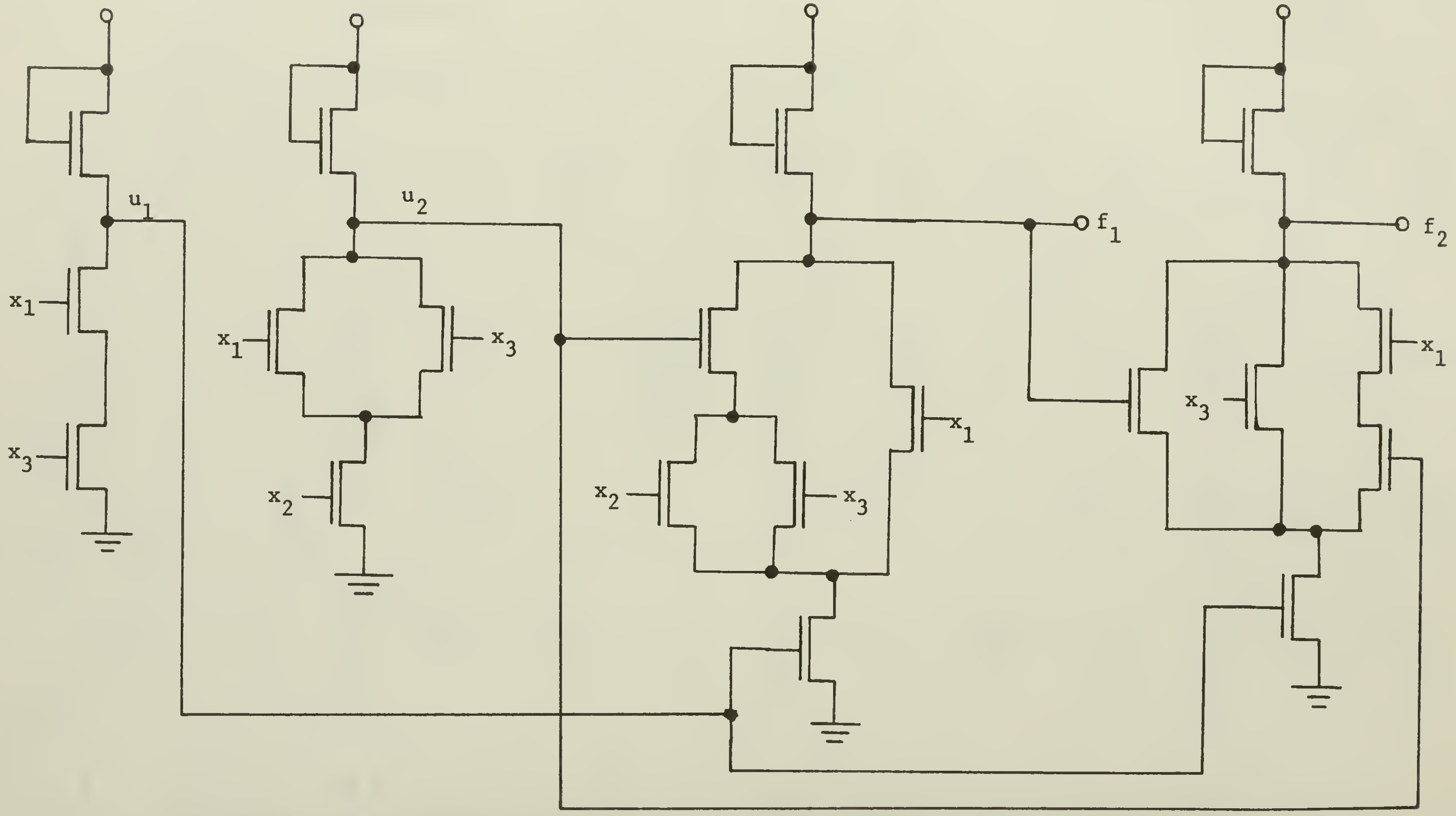
(ℓ) $\text{NFS}^2(4;\ f_1,f_2;\ 3,4)_3 = \text{NFS}(4;\ f_1,f_2;\ 3,4)_3 =$
$(u_1,u_2,\ f_1,f_2)$.

Fig. 9.1 (Continued)



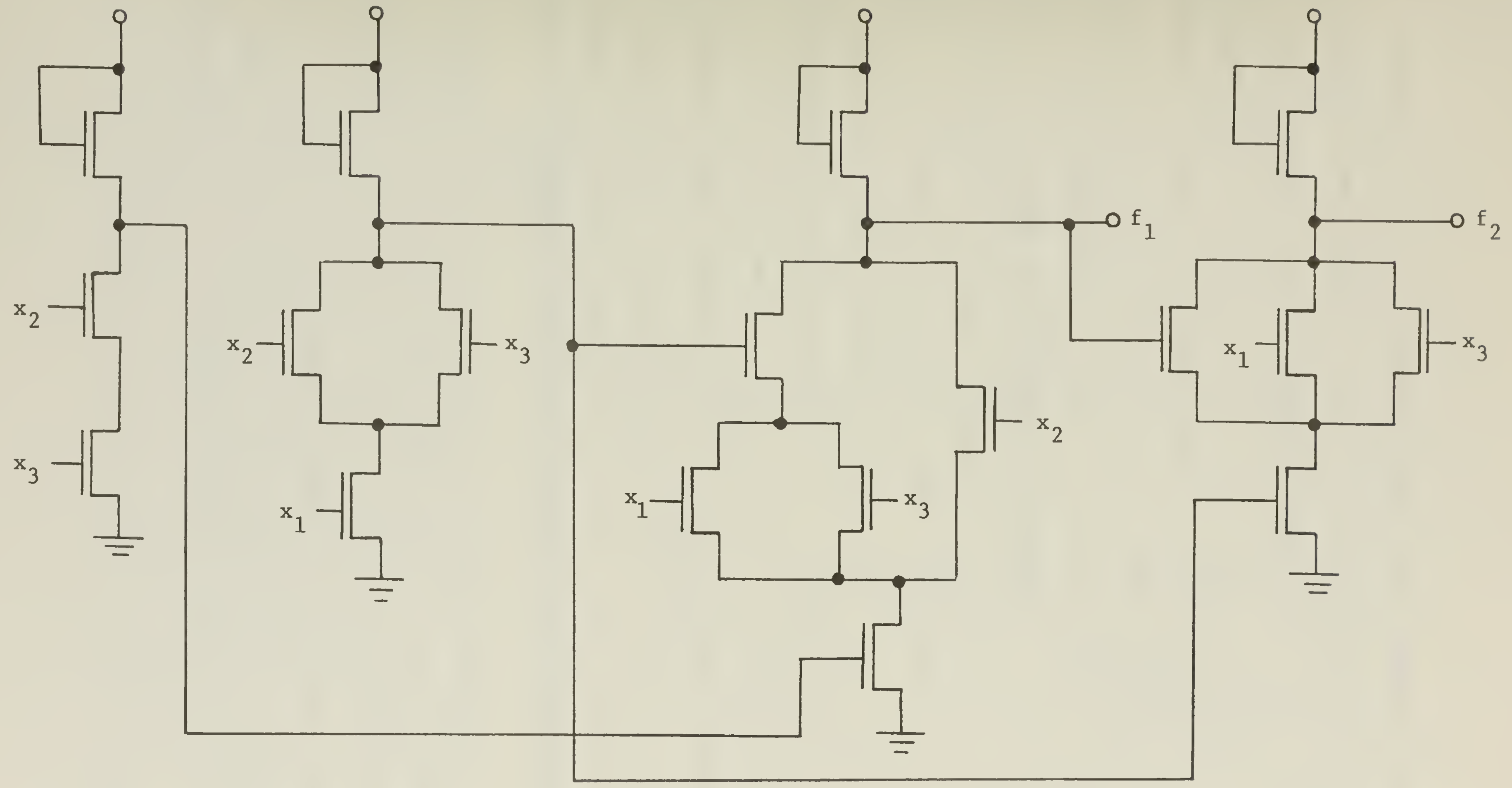
(m) Network corresponding to NFS(4; f_1,f_2; 3,4)$_1$ in (e).

<u>Fig. 9.1</u> (Continued)



(n) Network corresponding to NFS(4; f_1,f_2; 3,4)$_2$ in (i).

Fig. 9.1 (Continued)



(o) Network corresponding to NFS(4; f_1,f_2; 3,4)$_3$ in (ℓ).

<u>Fig. 9.1</u> (Continued)

least and the second least significant bits in L(A), respectively), the modified algorithm DIMN will obtain irredundant MOS networks of four gates. Fig. 9.1(b) shows both $\underline{NFS}^0(4; f_1,f_2; 3,4) = (\underline{u}_1, \underline{u}_2, f_1, f_2)$ and $\widehat{NFS}^0(4; f_1,f_2; 3,4) = (\hat{u}_1,\hat{u}_2, f_1,f_2)$. By comparing $\underline{u}_1$ and $\hat{u}_1$, NFS (4; f_1,f_2; 3,4) is first obtained (not shown), and then $NFS^1(4; f_1,f_2; 3,4)_1$, $NFS^1(4; f_1,f_2; 3,4)_2$, and $NFS^1(4; f_1,f_2; 3,4)_3$ are obtained as shown in (c), (f), and (j), respectively. Next, by Step 2 and Step 3 of the modified algorithm DIMN, $\underline{NFS}^1(4; f_1,f_2; 3,4)_i$ and $\widehat{NFS}^1(4; f_1,f_2; 3,4)_i$ are obtained for i=1,2,3, as shown in (d), (g), and (k), respectively. $\widetilde{NFS}^1(4; f_1,f_2; 3,4)_i$ and $NFS^2(4; f_1,f_2; 3,4)_i = NFS(4; f_1,f_2; 3,4)_i$ are then derived by Steps 4 and 5, as shown in (e), (h), (i), (ℓ). The irredundant networks corresponding to $NFS(4; f_1,f_2; 3,4)_i$ are shown in (m), (n), and (o) for i=1,2,3, respectively. It should be noted that a different irredundant MOS configuration for $\tilde{u}_2$ in $\widetilde{NFS}^1(4; f_1,f_2; 3,4)_3$ can be chosen instead of the one shown in Fig. 9.1(ℓ). This different choice will lead to a different MOS network which will be mentioned in Example 9.2.

It is sometimes desirable to realize all the output functions in the output level. In other words, gates which realize specified output functions are not allowed to feed other gates in the network. The generalized form of such networks which realize functions $f_1,\ldots, f_m$ and consist of R=r+m negative gates is shown in Fig. 9.2, where gates $g_{r+1},\ldots,g_{r+m}$ realize desired functions $f_1,\ldots,f_m$, respectively.

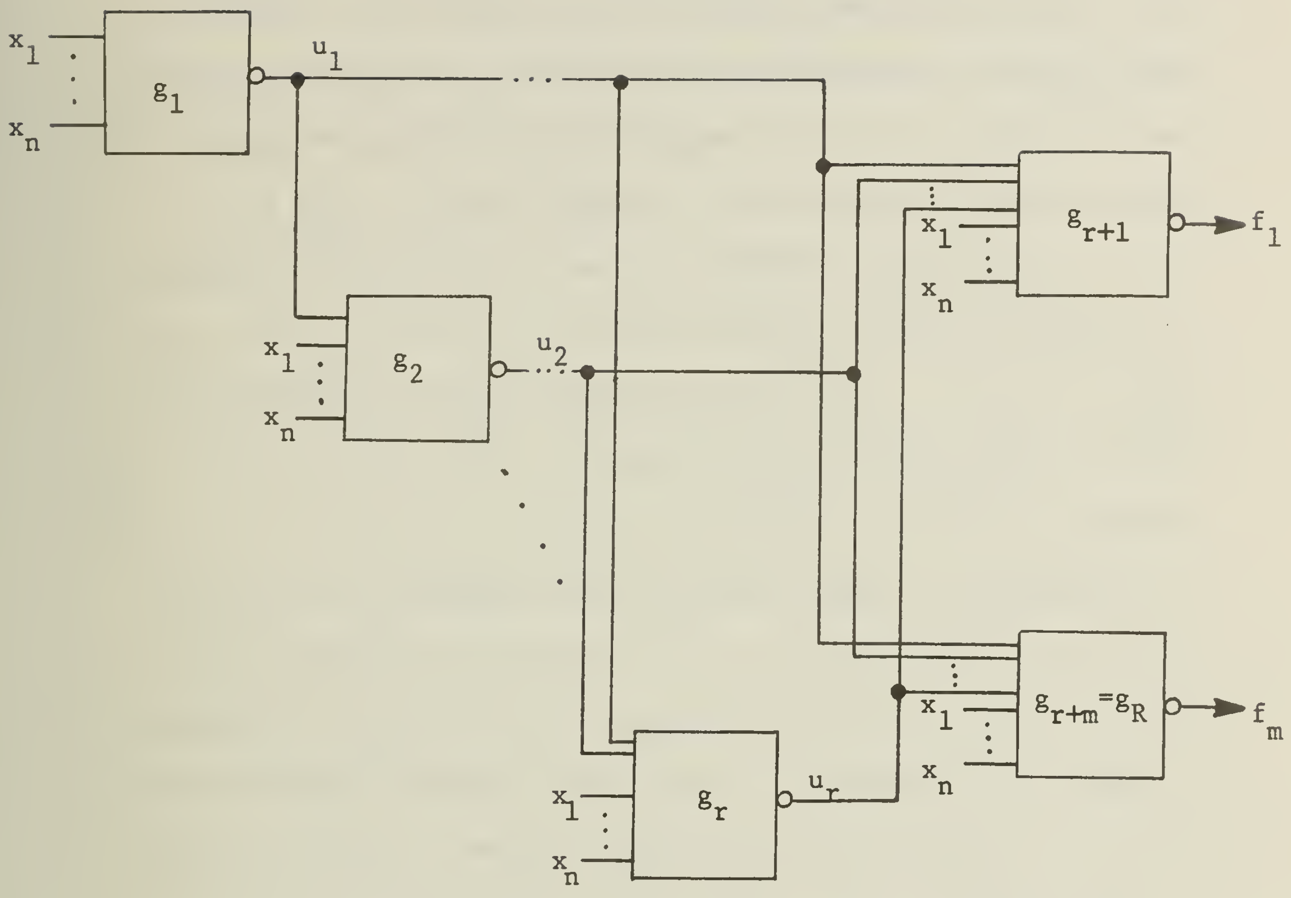


Fig. 9.2 Generalized form of networks which consist of R=r+m negative gates with all m output gates in the output level.

Some modifications of Algorithms 6.3.1 (CMNL), 6.3.2 (CMXL), and 6.4.1 (DIMN) are necessary in order to design irredundant MOS network in the form of Fig. 9.2 for a given set of functions, $f_1,\ldots,f_m$. Let us consider the subnetwork consisting of gates $g_1,\ldots,g_r,g_{r+1}$. This subnetwork is in the generalized form of networks consisting of r+1 negative gates shown in Fig. 3.1. Therefore, functions $(u_1,\ldots,u_r, f_1)$ constitute a negative function sequence of length r+1, i.e., NFS(r+1, f_1). Similarly, the sequence of functions $(u_1,\ldots,u_r, f_i)$ is also an NFS(r+1, f_i), for i=2,...,m. Based on this property, a network with a minimum number of negative gates in the form of Fig. 9.2 can be designed by the following modified algorithm MNL. For convenience, let NFS(r; m; $f_1,\ldots,f_m$) = $(u_1,\ldots,u_r; f_1,\ldots,f_m)$ represent a network in the form of Fig. 9.2, i.e., each $(u_1,\ldots,u_r, f_i)$ is an NFS(r+1, f_i) for i=1,...,m. Let $C_n(u_1,\ldots,u_r; f_1,\ldots,f_m)$ represent the corresponding labeled n-cube, where the label for vertex A is denoted by $\ell(A; u_1,\ldots,u_r; f_1,\ldots,f_m)$. It should be noted that the label consists of two parts: $(u_1,\ldots,u_r)$ is considered as a binary integer of r bits and $(f_1,\ldots,f_m)$ is considered as a vector of m bits. For two labels in vertices A and B, $\ell(A; u_1,\ldots,u_r; f_1,\ldots,f_m) = \ell(B; u_1,\ldots,u_r; f_1,\ldots,f_m)$ if and only if $u_i(A) = u_i(B)$ for i=1,...,r and $f_i(A) = f_i(B)$ for i=1,...,m; and $\ell(A; u_1,\ldots,u_r; f_1,\ldots,f_m) > \ell(B; u_1,\ldots,u_r; f_1,\ldots,f_m)$ if and only if either $\ell(A; u_1,\ldots,u_r) > \ell(B; u_1,\ldots,u_r)$, or $\ell(A; u_1,\ldots,u_r) = \ell(B; u_1,\ldots,u_r)$ and $(f_1(A),\ldots,f_m(A)) > (f_1(B),\ldots,f_m(B))$.

<u>Algorithm 9.1</u> - Minimum labeling for a given set of functions, $f_1,\ldots,f_m$ (MNL).

Let $L^{f_1,\ldots,f_m}(A)$ denote the first part of the label to be assigned to each vertex A ε C_n by this algorithm (i.e., $L_{mn}^{f_1,\ldots,f_m}(A) = \ell(A; u_1,\ldots,u_r)$).

<u>Step 1</u> - Assign $L_{mn}^{f_1,\ldots,f_m}(I) = 0$.

Step 2 - When $L_{mn}^{f_1,\ldots,f_m}(A)$ is assigned to each vertex of weight $w(1 \leq w \leq n)$ of C_n, assign as $L_{mn}^{f_1,\ldots,f_m}(B)$ to each vertex B of weight w-1 the smallest binary integer satisfying the following condition for each directed edge $\vec{AB} \in C_n$:

$$L_{mn}^{f_1,\ldots,f_m}(B) \geq L_{mn}^{f_1,\ldots,f_m}(A) \text{ if } (f_1(A),\ldots,f_m(A)) \leq (f_1(B),\ldots,f_m(B)),$$

$$L_{mn}^{f_1,\ldots,f_m}(B) > L_{mn}^{f_1,\ldots,f_m}(A) \text{ if } (f_1(A),\ldots,f_m(A)) \not\leq (f_1(B),\ldots,f_m(B))^{\dagger}.$$

Step 3 - Repeat Step 2 until $L_{mn}^{f_1,\ldots,f_m}(0)$ is assigned.

Step 4 - The number of bits in $L_{mn}^{f_1,\ldots,f_m}(0)$ plus m is the minimum number of negative gates required to realize $f_1,\ldots,f_m$ in the form of Fig. 9.2. The i-th most significant bit of $L_{mn}^{f_1,\ldots,f_m}(A)$ is denoted by $u_i(A)$ for each $A \in C_n$ and $i=1,\ldots,R$. $(u_1,\ldots,u_R; f_1,\ldots,f_m) = NFS(r+m; f_1,\ldots,f_m)$ has been obtained as the negative function sequence with multiple-outputs.

The validity of this algorithm can be proved based on that $C_n(u_1,\ldots,u_r, f_i)$ has no inverse edge for $i=1,\ldots,m$ because the condition in Step 2 is satisfied and that $L_{mn}^{f_1,\ldots,f_m}(A)$ is chosen

---

†Since this is a relation between two vectors, $(f_1(A),\ldots, f_m(A)) \not\leq (f_1(B),\ldots,f_m(B))$ means that either $(f_1,\ldots,f_m^1(A)) > (f_1(B),\ldots,f_m(B))$ or they are incomparable.

so that the total number of bits in $L_{mn}^{f_1,\ldots,f_m}(0)$ is minimum and hence $R = r+m$ is the minimum number of negative gates to realize $f_1,\ldots,f_m$ in the form of Fig. 9.2.

Algorithms 6.3.1 (CMNL) and 6.3.2 (CMXL) can be similarly modified. Given a partially specified negative function sequence $NFS^i(r+m;\ f_1,\ldots,f_m) = (u_1,\ldots,u_i,u^*_{i+1},\ldots,u^*_r;\ f_1,\ldots,f_m)$, algorithm CMNL assigns to each vertex $A \in C_n$ a minimum possible label $L_{mn}^{f_1,\ldots,f_m;i}(A) = \ell(A;\ u_1,\ldots,u_i,\ \underline{u}_{i+1},\ldots,\underline{u}_r)$ satisfying that

(a) The k-th bit of $L_{mn}^{f_1,\ldots,f_m;i}(A)$ is $u_k(A)$

(b) $\ell(B;\ u_1,\ldots,u_i,\underline{u}_{i+1},\ldots,\underline{u}_r;\ f_1,\ldots,f_m) \geq \ell(A;\ u_1,\ldots,u_i,\ \underline{u}_{i+1},\ldots,\underline{u}_r;\ f_1,\ldots,f_m)$ for each directed edge $\vec{AB} \in C_n$.

Similarly, the modified algorithm CMXL assigns a maximum possible label $L_{mx}^{f_1,\ldots,f_m;i}(A) = \ell(A;\ u_1,\ldots,u_i,\hat{u}_{i+1},\ldots,\hat{u}_r)$ to each vertex $A \in C_n$ satisfying the same conditions. Based on these two modified algorithms, Algorithm 6.4.1 (DIMN) then can be modified as follows.

<u>Algorithm 9.2</u> - Design of irredundant MOS networks for a given set of functions $f_1,\ldots,f_m$ (DIMN).

<u>Step 1</u> - Let $NFS^o(r+m;\ f_1,\ldots,f_m) = (u^*_1,\ldots,u^*_r;\ f_1,\ldots,f_m)$ and i=1. (If r is not known at this step, it will be determined after <u>NFS</u>$^o(r+m;\ f_1,\ldots,f_m)$ is obtained in Step 2 by applying the MNL.)

<u>Step 2</u> - Using the modified algorithm CMNL, obtain $NFS^{i-1}(r+m;\ f_1,\ldots,f_m) = (u_1,\ldots,u_{i-1},\ \underline{u}_i,\ldots,\underline{u}_r;\ f_1,\ldots,f_m)$.

Step 3 - Using the modified algorithm CMXL, obtain $\widehat{NFS}^{i-1}(r+m;\ f_1,\ldots,f_m) = (u_1,\ldots,u_{i-1},\ \hat{u}_i,\ldots,\hat{u}_r;\ f_1,\ldots,f_m)$.

Step 4 - Obtain function $\tilde{u}_i$ satisfying

$\tilde{u}_i(A) = 0$, if $\underline{u}_i(A) = \hat{u}_i(A)=0$;

$\tilde{u}_i(A) = 1$, if $\underline{u}_i(A) = \hat{u}_i(A)=1$; and

$\tilde{u}_i(A) = *$, if $\underline{u}_i(A)=0$ and $\hat{u}_i(A)=1$.

(The combination of $\underline{u}_i(A)=1$ and $\hat{u}_i(A)=0$ can never occur.)

Step 5 - Obtain an irredundant MOS cell configuration for $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_{i-1}$ by an appropriate algorithm (e.g., Algorithms 5.1 or 5.2). Let u_i be the function realized by this cell (u_i is a completion of $\tilde{u}_i$ with respect to $x_1,\ldots,x_n$).

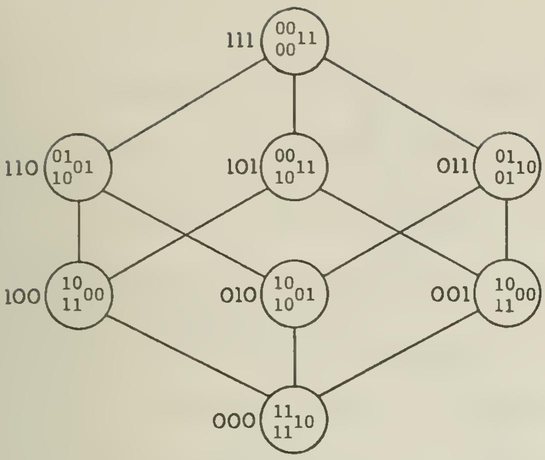
Step 6 - If i=r, design an irredundant MOS cell configuration for each f_j with respect to $x_1,\ldots,x_n$, $u_1,\ldots,u_r$ for $j=1,\ldots,m$, and terminate this algorithm; otherwise set $i=i+1$ and go to Step 2.

For proving that this algorithm designs irredundant MOS networks in the form of Fig. 9.2 for $f_1,\ldots,f_m$, the property of $\tilde{u}_i$ plays an important role. Based on the discussion in Section 6, it is not difficult to understand that $(u_1,\ldots,u_{i-1},\ \overset{o}{u}_i,\ \overset{*}{u}_{i+1},\ldots,\overset{*}{u}_r;\ f_1,\ldots,f_m)$ will not be a feasible partially specified $NFS^i(r+m;\ f_1,\ldots,f_m)$ if $\overset{o}{u}_i$ is not a completion of $\tilde{u}_i$. In other words, if any FET is extracted from cell g_i, no network in the form of Fig. 9.2 can be designed no matter how $\overset{*}{u}_{i+1},\ldots,\overset{*}{u}_r$ are specified. The detailed proof is omitted.

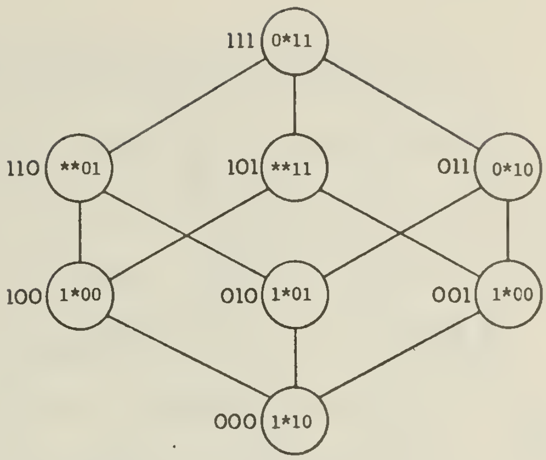
The diagnostic properties of the networks designed by Algorithm 9.2 for multiple output functions are the same as discussed earlier in this section.

Example 9.2 - Consider the set of functions f_1 and f_2 discussed in Example 9.1. All three solutions given in Example 9.1 require a connection from gate g_3 to gate g_4, i.e., the gate realizing function f, is not in the output level. Here, let us design an irredundant MOS network for f_1 and f_2 with the gates realizing the two both in the output level. Fig. 9.3(a) shows $\underline{NFS}^o(4;\ f_1,f_2)$ and $\widehat{NFS}^o(4;\ f_1,f_2)$ obtained by Steps 2 and 3 of Algorithm 9.2 (DIMN), respectively. Comparing $\underline{u}_1$ and $\hat{u}_1$ in Step 4, $\tilde{u}_1$ is obtained, and hence $\widetilde{NFS}^o(4;\ f_1,f_2) = (\tilde{u}_1,u_2^*;\ f_1,f_2)$ is obtained as shown in (b). The only irredundant MOS cell configuration for $\tilde{u}_1$ is $u_1 = \overline{x_2x_3}$ (obtained by Step 5), and the corresponding $NFS^1(4;\ f_1,f_2) = (u_1,u_2^*;\ f_1,f_2)$ is shown in (c). Fig. 9.3(d) shows both $\underline{NFS}^1(4;\ f_1,f_2) = (u_1,\underline{u}_2;\ f_1,f_2)$ and $\widehat{NFS}^1(4;\ f_1,f_2) = (u_1,\hat{u}_2;\ f_1,f_2)$ obtained by Steps 2 and 3, respectively. Since $\underline{u}_2(A) = \hat{u}_2(A)$ for all $A \in C_3$; $\tilde{u}_2 = \underline{u}_2 = \hat{u}_2$ is completely specified, and the solution $NFS(4;\ f_1,f_2)$ is obtained as shown in Fig. 9.3(e). The network corresponding to this solution is obtained by designing irredundant MOS cell configurations for u_2 (Step 5), f_1, and f_2 (Step 6) and is shown in (f).

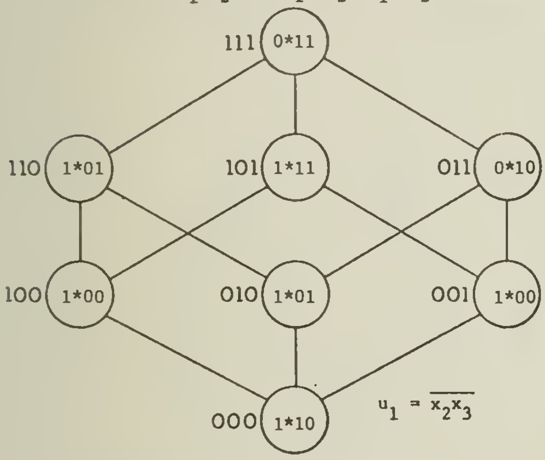
It is interesting to see that the network in Fig. 9.3(f) consists of only 17 FETs while the networks shown in Fig. 9.1(m), (n) and (o) consists of 19, 20, and 18 FETs, respectively. In general,



(a) $\underline{NFS}^{\circ}(2;\ f_1,f_2) = (\underline{u}_1,\ \underline{u}_2,\ f_1,\ f_2)$.
$\widehat{NFS}^{\circ}(2;\ f_1,f_2) = (\hat{u}_1,\ \hat{u}_2,\ f_1,\ f_2)$.

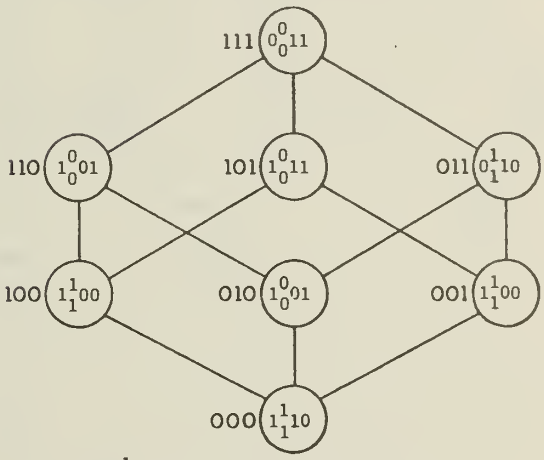


(b) $\tilde{NFS}^{\circ}(2;\ f_1,f_2) = (\tilde{u}_1,\ u_2^{*};\ f_1,f_2)$.

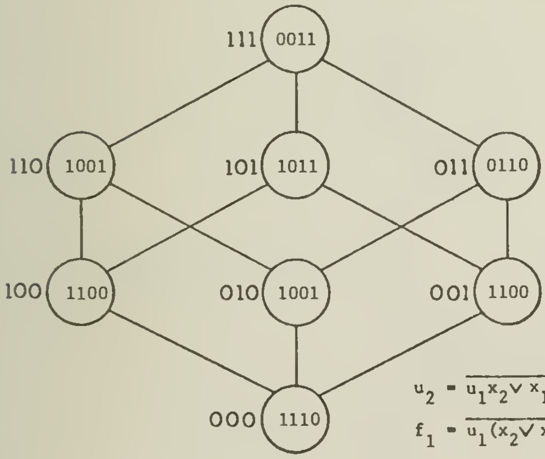


$u_1 = \overline{x_2x_3}$

(c) $NFS^1(2;\ f_1,f_2) = (u_1,\ u_2^{*};\ f_1,\ f_2)$.



(d) $\underline{NFS}^1(2;\ f_1,f_2) = (u_1,\underline{u}_2;\ f_1,f_2)$.
$\widehat{NFS}^1(2;\ f_1,f_2) = (u_1,\hat{u}_2;\ f_1,f_2)$.



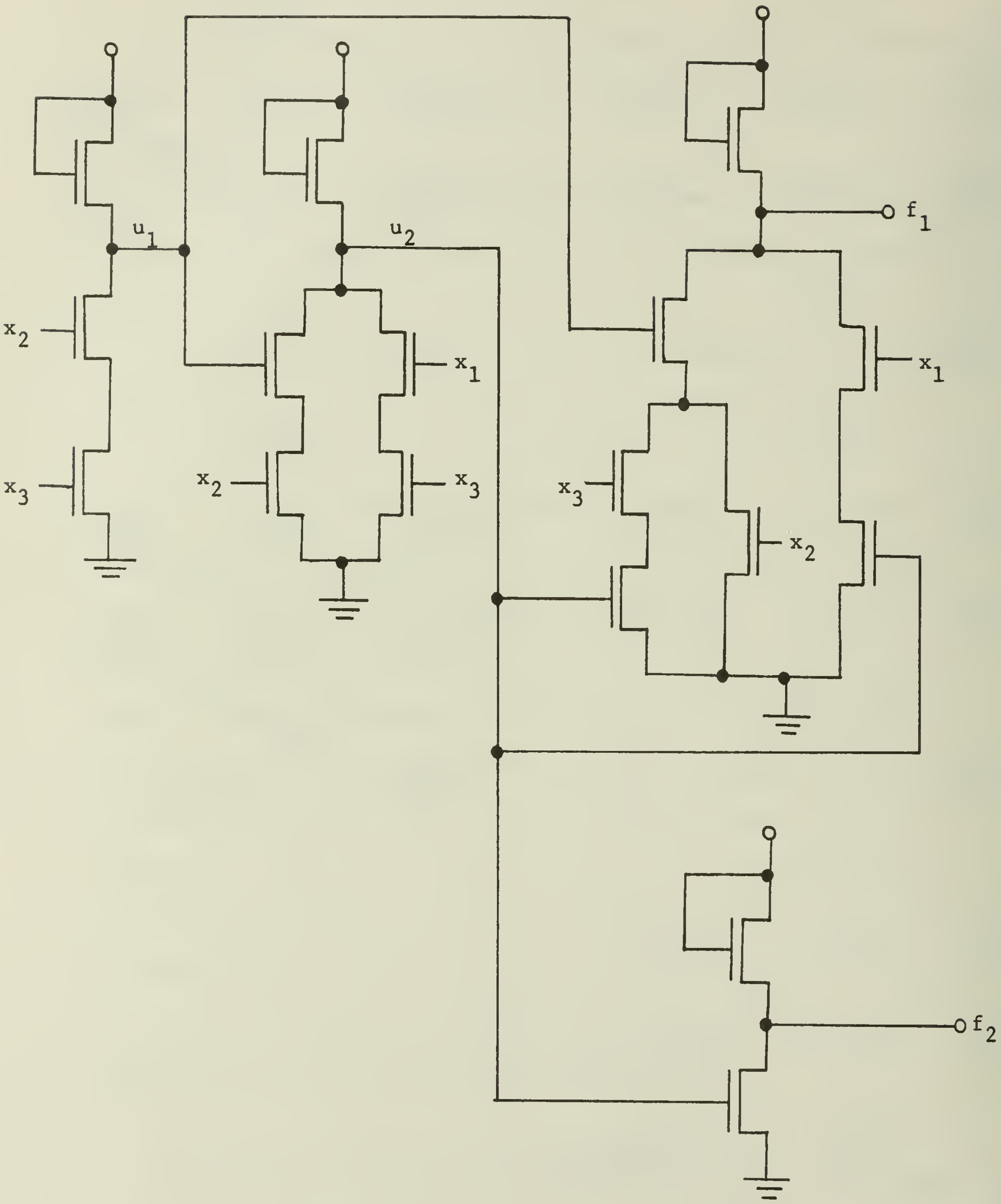
$u_2 = \overline{u_1x_2 \vee x_1x_3}$

$f_1 = \overline{u_1(x_2 \vee x_3u_2) \vee x_1u_2}$

$f_2 = \overline{u}_2$

(e) $\widehat{NFS}^1(2;\ f_1,f_2) = NFS^2(2;\ f_1,f_2) = NFS(2;\ f_1,f_2)$.

Fig. 9.3 Example 9.2.



(f) Network corresponding to NFS(2; f_1,f_2) of (e).

Fig. 9.3 (Continued)

however, the networks designed by Algorithm 9.2 for designing irredundant MOS networks with all output gates in the output level tend to contain more FETs than those without such restrictions since they consist of no less number of MOS cells than those networks without such restrictions. Another interesting observation is that Example 9.1 can also obtain the network in Fig. 9.3(f) from $\underline{NFS}^1(4; f_1,f_2; 3,4)_3$ and $\widehat{NFS}^1(4; f_1,f_2; 3,4)_3$ of Fig. 9.1(k), though the network was not shown in Fig. 9.1.

10. CONCLUSIONS

An algorithm for the design of irredundant MOS networks for single or multiple, completely or incompletely specified functions has been presented in the previous sections. For a given single function, the algorithm designs an irredundant MOS network with a minimum number of MOS cells (negative gates). For a given set of more than one function, on the other hand, the number of MOS cells in the network designed by this algorithm may not be the minimum required. In both cases, however, the designed networks are free of redundant FETs (i.e., irredundant) and therefore diagnosable for all single and multiple stuck-at-short and/or stuck-at-open faults.

The algorithm first designs the MOS cell in the network remotest from the output to be as simple as possible so that any extractions of FETs from that cell will make the rest of the network unrealizable without the addition of extra cells. Based on the first cell and the given output function(s), the algorithm then designs the second cell to be as simple as possible and so forth until all cells are completely designed. Because of the manner in which the network is designed, an MOS cell with a smaller subscript (i.e., remoter from the output cell) tends to be simpler in structure than one with a larger subscript. This means that the cells tend to have unequal complexity, making the cells close to the output cell more complex than those relatively far away from the output cell. This property is sometimes undesirable because a complex cell may create difficulties in the design of the layout of the cell and also may have an adverse effect upon the speed of the entire network. Due to this disadvantage, other methods capable of designing MOS networks under certain restrictions, such as restrictions on the numbers of FETs in series and in parallel within an MOS cell, are more desirable in practice. This problem is left for another occasion.

The diagnostic properties of the networks designed by the algorithms discussed in Part I of the thesis are very useful especially when the given function is sparsely specified (i.e., has a relatively large number of don't care components).

Examples 6.5.1 and 8.1 illustrate an interesting problem. For the completely specified function f in Example 6.5.1 and the incompletely specified function $\tilde{f}$ in Example 8.1, algorithm DIMN designs identical networks. Therefore, the set of input vectors specified for $\tilde{f}$ is a sufficient test set of the networks designed for f. If we can find a relatively simple way to obtain an incompletely specified function $\tilde{f}$ from a completely specified function f such that algorithm DIMN designs an identical network for both f and $\tilde{f}$, then we will have a simple method to obtain a sufficient test set for the network designed for f which has a fewer number of input vectors. This problem is also left for another occasion.

It should be understood that the algorithms can be applied to the logical design with logical devices other than MOSFETs as long as the logical devices to be used can realize an arbitrary negative function. A good example is AOI (AND-OR-INVERT) gate which is widely used in practice. Fig. 10.1 shows the logic diagram of such a gate.

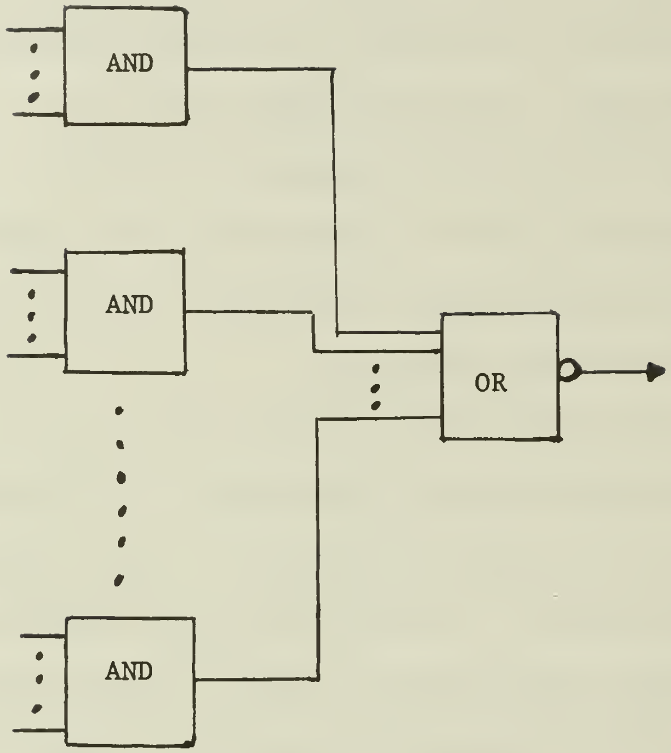


Fig. 10.1 An AND-OR-INVERT gate.

If there is no fan-in restriction on the number of inputs to each AND element and the NOR element, an AOI gate can realize an arbitrary negative function, and therefore algorithm DIMN can be applied to the design or irredundant networks consisting of AOI gates. In such an application, an algorithm to obtain complemented irredundant disjunctive forms for a given incompletely specified function (e.g., Algorithm 5.1) should be used since an AOI gate corresponds to a complemented disjunctive form. A network designed by this approach is guaranteed to be irredundant--which means all single and multiple stuck-at-0 and/or stuck-at-1 faults in the network can be detected by examining the output of the network only.

REFERENCES

[Ake 73] S. R. Akers, Jr., "Universal test sets for logic networks," IEEE Trans. Computers, vol. C-22 no. 9, pp. 835-839, Sept. 1973.

[Gil 54] E. N. Gilbert, "Lattice theoretic properties of frontal switching functions," J. Math. Phys., vol. 33, pp. 57-67, April 1954.

[Iba 71] J. Ibaraki, "Gate-interconnection minimization of switching networks using negative gates," IEEE Trans. Computers, vol. C-20, pp. 698-706, June 1971.

[IM 71] T. Ibaraki and S. Muroga, "Synthesis of networks with a minimum number of negative gates," IEEE Trans. Computers, vol. C-20, pp. 49-58, Jan. 1971.

[Law 64] E. L. Lawler, "An approach to multilevel Boolean minimization," J. ACM. vol. 11, no. 3, pp. 283-295, July 1964.

[Liu 72] T. K. Liu, "Synthesis of logic networks with MOS complex cells," Report UIUCDCS-R-72-517, Dept. of Comp. Sci., Univ. of Ill., Urbana, Ill., May 1972.

[Mar 58] A. A. Markov, "On the inversion complexity of a system of functions," J. ACM, vol. 5, pp. 331-334, Oct. 1958.

[Mor 72] Y. Moriwaki, "Synthesis of minimum contact networks based on Boolean polynomials and its programming on a digital computer," Report of Institute of Industrial Science, vol. 21, no. 6, University of Tokyo, Tokyo, Japan, Mar. 1972.

[Mul 58] D. E. Muller, "Minimizing the number of NOT elements in combinational circuits," Memorandum, Bell Telephone Laboratories, 1958.

[NTK 72] K. Nakamura, N. Tokura, and T. Kasami, "Minimal negative gate networks," IEEE Trans. Computers, vol. C-21, pp. 5-11, Jan. 1972.

[Pai 73] M. R. Paige, "Synthesis of diagnosable FET networks," IEEE Trans. Computers, vol. C-22, no. 5, May 1973.

[SK 69] G. L. Schnable and R. S. Keen, Jr., "Failure mechanisms in large-scale integrated circuits," IEEE Trans. Electron Devices, vol. ED-16, pp. 322-332, April 1968.

[Spe 69] R. F. Spencer, Jr., "Complex gates in digital system design," IEEE Computer Group News, pp. 47-56, Sept. 1969.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-79-996	2.	3. Recipient's Accession No.

. Title and Subtitle
ESIGN OF DIAGNOSABLE MOS NETWORKS

5. Report Date
Dec. 1979

6.

. Author(s)
Hung Chi Lai

8. Performing Organization Rept. No. UIUCDCS-R-79-996

. Performing Organization Name and Address
epartment of Computer Science
niversity of Illinois
rbana, IL 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.
NSF Grant No. DCR73-03421 A01

2. Sponsoring Organization Name and Address
ational Science Foundation
800 G. Street, N.W.
ashington, D.C.

13. Type of Report & Period Covered
Technical Report

14.

5. Supplementary Notes

6. Abstracts
This paper presents an algorithm designing multilevel MOS networks with a minimum umber of MOS cells and irredundant interconnections for a given switching function. his network is diagnosable. Then this algorithm is extended to the case of networks ith incompletely specified multiple output functions, although the minimality of the umber of cells is not guaranteed. Some diagnostic properties of the designed etworks are also discussed.

7. Key Words and Document Analysis. 17a. Descriptors
ogic design, minimal networks, minimum no. of MOS cells, irredundant interconnections, OS networks, diagnosis

7b. Identifiers/Open-Ended Terms

7c. COSATI Field/Group

8. Availability Statement RELEASE UNLIMITED	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 177
	20. Security Class (This Page UNCLASSIFIED	22. Price